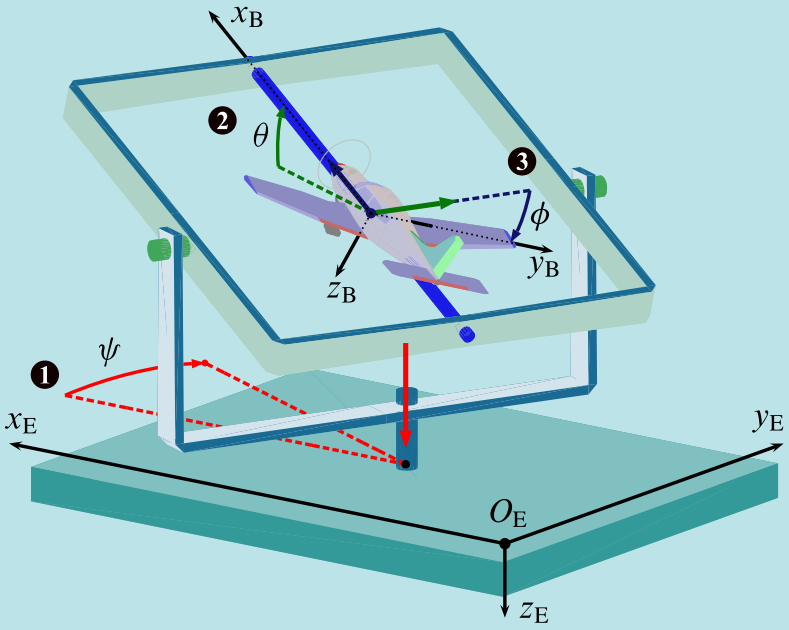

Flight Dynamics with Matlab/Simulink

Elaborato di Dinamica e simulazione di volo

Part I

Antonio Carotenuto

A.A 2023-2024

Indice

Elenco delle figure

Capitolo 1

Aircraft Kinematics

1.1 Introduction

This exercise provides a kinematic description of the motion of an aircraft during several maneuvers. The following hypotheses are made a priori:

• Aircraft assumed as a rigid body: aeroelastic effects are neglected, shape variation due to the deflection of mobile surfaces is not considered.

• It is assumed that the reference frame with origin at a point on the Earth's surface and axes oriented according to the NED convention is an inertial reference frame and that the Earth is flat. These hypotheses are considered valid with good approximation because the phenomena studied are characterized by short times and low speeds.

The position and attitude of the aircraft are defined using the position vector $\{x_{EG}, y_{EG}, z_{EG}\}$ and the Euler angles $\{\phi, \theta, \psi\}$ which identify the orientation of the reference frame fixed to the aircraft itself. As an alternative to Euler angles, the orientation quaternion can be used, a representation that offers a more efficient and singularity-free way to describe the aircraft's orientation during maneuvers, providing a solid mathematical basis for the kinematic analysis of motion.

1.2 Problem Setup

1.2.1 Formulation in terms of Euler angles

In scientific literature, Euler angles are one of the most used ways to represent the orientation in space of a mobile frame with respect to a fixed frame. Remaining within our area of interest, we consider a fixed frame (Earth-fixed frame assumed inertial) and a mobile frame (Body frame fixed to the aircraft). Euler angles define 3 consecutive and ordered rotations that a fictitious reference, initially oriented like the fixed frame, must perform to overlap with the mobile frame. In particular, the sequence of rotations we will examine will be 321 (or ZYX). In this way, we can define transformation laws for the components of a vector. In our case, we are interested in expressing the coordinates of a

vector, for example, in the Earth frame starting from the knowledge of Euler angles and its coordinates in the Body frame. The rotation matrix that allows us to do this is the following:

$$\begin{bmatrix} C\theta C\phi & C\theta S\psi & -S\theta \\ S\phi S\theta C\psi - C\phi S\psi & S\phi S\theta S\psi + C\phi C\psi & S\phi C\theta \\ C\phi S\theta C\psi + S\psi S\theta & C\phi S\theta S\psi - S\theta C\psi & C\theta C\phi \end{bmatrix} \tag{1.1}$$

This matrix is the product of the matrices corresponding to the individual rotations mentioned above. They possess the property of orthogonality (inverse = transpose). Transformation matrices are also called direction cosine matrices (DCM). By applying the transformation matrices from the knowledge of the velocity vector in the body frame, we can derive the time derivatives of the position vector in the inertial frame:

$$\begin{Bmatrix} \dot{x}_{E,G} \\ \dot{y}_{E,G} \\ \dot{z}_{E,G} \end{Bmatrix} = \begin{vmatrix} C\theta C\phi & C\theta S\psi & -S\theta \\ S\phi S\theta C\psi - C\phi S\psi & S\phi S\theta S\psi + C\phi C\psi & S\phi C\theta \\ C\phi S\theta C\psi + S\psi S\theta & C\phi S\theta S\psi - S\theta C\psi & C\theta C\phi \end{vmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{1.2}$$

The equation just mentioned is called *navigation equations* . Using the transformation laws of the components of a vector from one reference frame to another, it is possible to establish the relationship that exists between the instantaneous angular velocity vector in body axes (p,q,r) and the time derivatives of the Euler angles. Based on what has been said about Euler angles, the following relationships exist:

$$\begin{Bmatrix} p \\ q \\ r \end{Bmatrix} = \begin{bmatrix} 1 & 0 & -S\theta \\ 0 & C\phi & S\phi C\theta \\ 0 & -S\phi & C\phi C\theta \end{bmatrix} \begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} \tag{1.3}$$

$$\begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} 1 & \frac{S\phi S\theta}{C\theta} & \frac{C\phi S\theta}{C\theta} \\ 0 & C\phi & -S\phi \\ 0 & \frac{S\phi}{C\theta} & \frac{C\phi}{C\theta} \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \tag{1.4}$$

The equation just mentioned is called *Gimbal equations*. The Gimbal equations together with the Navigation equations form a system of 6 coupled equations, called *Auxiliary Kinematic Equations*. Knowing the time derivatives of the Euler angles and the components of instantaneous velocity in the aircraft reference frame, and given the initial conditions, the differential problem is well-posed. By integrating, we can obtain the time history of the Euler angles and the coordinates of the center of gravity in the assumed inertial frame (and therefore the attitude and position of the aircraft instant by instant). Ultimately, through the knowledge of the 3 coordinates of the center of gravity and the 3 Euler angles, we have saturated the 6 degrees of freedom in space of a rigid body, and the kinematics is known. It should be noted, however, that the Gimbal equations have a singularity point, and for this reason, it is advisable to use a different parameterization of the orientation.

1.2.2 Formulation in terms of quaternion

Clearly, as specified above, the aircraft dynamics are described by 3 translational and 3 rotational degrees of freedom, but it is entirely permissible to use 4 parameters to cover

the 3 rotational degrees of freedom provided that one of them is dependent on the other 3. We recall Euler's axis theorem:

Theorem 1: *The instantaneous orientation of a rigid body, whose motion is constrained to rotations around a fixed point, can always be described as that originating from a single rotation from the initial orientation, by a given angle and around a given axis.*

The axis around which the rotation must occur passes through the hinge (in our case, the center of gravity), and we denote the unit vector of this axis by e. It will clearly be characterized by 3 components; together with the angle by which the frame must rotate (which we denote by μ), we have the 4 components. However, since e is a unit vector, it necessarily follows that $e_x^2 + e_y^2 + e_z^2 = 1$. Through a change of variables known as *Euler-Rodrigues parameterization*, we can link the formulation that uses Euler's axis to the one that describes motion with the set of Euler angles:

$$\left\{ \begin{array}{c} q_0 \\ q_x \\ q_y \\ q_z \end{array} \right\} = \left\{ \begin{array}{c} \cos\frac{\mu}{2} \\ e_x \sin\frac{\mu}{2} \\ e_y \sin\frac{\mu}{2} \\ e_z \sin\frac{\mu}{2} \end{array} \right\} \tag{1.5}$$

Through manipulations of the *Euler-Rodrigues parameters* and their relationships with the angular velocity in body axes, one arrives at the *gimbal equation* in terms of the orientation quaternion:

$$\left\{ \begin{array}{c} \dot{q}_0 \\ \dot{q}_x \\ \dot{q}_y \\ \dot{q}_z \end{array} \right\} = \frac{1}{2} \left[\begin{array}{cccc} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{array} \right] \left\{ \begin{array}{c} q_0 \\ q_x \\ q_y \\ q_z \end{array} \right\} \tag{1.6}$$

After defining the time history of the vector Ω_b, it is possible to numerically integrate (in our treatment, the ode45 command will be used, an embedded numerical integration method based on fourth and fifth-order *Runge-Kutta* schemes) the system of four first-order ordinary differential equations. In this way, we obtain the components of the quaternion **q** by assigning appropriate initial conditions. It remains to determine a unique and general relationship that allows obtaining Euler Angles from orientation quaternions and vice versa. It may be necessary to determine the quaternion components given the Euler angles if, for example, the initial conditions are expressed in terms of Euler angles. It may be necessary to convert from quaternion components to Euler angles if, for example, one wants to create a plot of the trajectory. This operation is trivial in *Matlab* using the *quat2angle* function, which internally implements the following transformation:

$$\left\{ \begin{array}{c} \phi \\ \theta \\ \psi \end{array} \right\} = \left\{ \begin{array}{c} \text{atan}\,2\left[2\left(q_0 q_x + q_y q_z\right), \left(q_0^2 - q_x^2 - q_y^2 - q_z^2\right)\right] \\ \text{asin}\left[2\left(q_0 q_y\right) - \left(q_x q_z\right)\right] \\ \text{atan}\,2\left[2\left(q_0 q_z + q_x q_y\right), \left(q_0^2 + q_x^2 - q_y^2 - q_z^2\right)\right] \end{array} \right\} \tag{1.7}$$

1.3 Kinematics Exercises

In this notebook, the task was to model maneuvers by assigning the evolution laws of the kinematic motion variables in terms of linear and angular velocity. Specifically, through 1D interpolation, time functions are constructed that express the laws of [u, v,w, p, q, r](t) to model the motion. Given the time histories and appropriate initial conditions in terms of initial position in an inertial reference system (Earth fixed) and attitude in terms of orientation quaternion $\{q_0, q_x, q_y, q_z\}$ or Euler angles $\{\phi, \theta, \psi\}$, the Gimbal equations are integrated to obtain the attitude instant by instant. After that, the transformation matrix $\{T_{EB}\}$ is calculated, which is a function instant by instant of the attitude, and the navigation equations are integrated to obtain the time history of the center of gravity position $\{x_{EG}, y_{EG}, z_{EG}\}$.

1.3.1 Kinematics of the Cobra Maneuver

The cobra maneuver is a demanding and spectacular aerial acrobatics, practicable only and exclusively with some modern fighter aircraft. The maneuver starts in level flight at moderate speed when, with the angle of attack limiter disengaged (the angle formed by the aircraft's direction with respect to the airflow investing it), the pilot abruptly raises the aircraft's nose exceeding 100° of angle of attack, and then returns, more gently, to level flight. Through the thrust of powerful engines, the jet maintains an approximately constant altitude. Below is the code to integrate the auxiliary kinematic equations in the case of initial conditions and laws for instantaneous velocity and angular velocity relative to the cobra motion.

Listing 1.1

```
psi0 = 0; theta0 = 0; phi0 = 0; % assi body orientato come assi Earth
Z0 = 0;

%% Anonymous functions

% LOOPING
% three appropriate time histories of p, q, r
p_max = convangvel(0.0,'deg/s','rad/s'); % rad/s
q_max = convangvel(11,'deg/s','rad/s'); % rad/s
r_max = convangvel(0,'deg/s','rad/s'); % rad/s

% tempo di simulazione
t_fin=28;

p = @(t) 0*t;

%velocit  angolari
p = @(t) 0*t;

vBreakPointsQ(1,:) = [0 , 0.0179 , 0.0357 , 0.0714 , 0.1071 , 0.2857 ,...
                      0.3571 , 0.4643 , 0.7857 , 1.0714]*t_fin;

vBreakPointsQ(2,:) = [0 , 0.3907 , 0.9636 , 1.0417 , 2.1095 , 0.9063 ,...
                     -0.8292 , -1.4584 , -0.1042 , -0.0156]*q_max;
q = @(t) ...
```

```
    interp1( vBreakPointsQ(1,:), vBreakPointsQ(2,:), t, 'pchip' );

r = @(t) 0*t;

% three appropriate time histories of u, v, w
u0 = convvel(180 , 'km/h','m/s');
w0 = convvel(0 , 'km/h','m/s');

vBreakPointsU(1 ,:) = [0 , 0.0714 , 0.1786 , 0.3571 , 0.5 , 0.6429 ,...
                       0.8929 , 1.0714]*t_fin;
vBreakPointsU(2 ,:) = [1 , 0.5 , 0.28 , 0.01 , 0.2 , 0.4 ,...
                       0.8 , 1.3]*u0;

u = @(t) ...
    interp1( vBreakPointsU(1,:), vBreakPointsU(2,:), t, 'pchip' );
% assume alpha = beta = 0
v = @(t) ...
    0;
w = @(t) ...
    0;

%% Anonymous functions
% RHS of quaternion components evolution equations, see eq. (2.27)
dQdt= @(t,Q) 0.5.*....
[ 0 -p(t) -q(t) -r(t) ;
p(t) 0 r(t) -q(t) ;
q(t) -r(t) 0 p(t) ;
r(t) q(t) -p(t) 0 ]*Q;

%% Solution of Gimbal equations
options = odeset( ...
    'RelTol', 1e-9, ...
    'AbsTol', 1e-9 ...
    );

Q0=angle2quat(psi0, theta0, phi0);

[vTime, vQ] = ode45(dQdt, [0 t_fin], Q0, options);


[Psi , Theta , Phi] = quat2angle(vQ);

vPhiThetaPsi=[Phi,Theta,Psi];


%% Integrate Navigation equations
% Time interpolation function for known Euler angles histories
fPhi = @(t) ...
    interp1(vTime,vPhiThetaPsi(:,1),t);
fTheta = @(t) ...
    interp1(vTime,vPhiThetaPsi(:,2),t);
fPsi = @(t) ...
    interp1(vTime,vPhiThetaPsi(:,3),t);
% RHS of navigation equations
dPosEdt = @(t,Pos) ...
```

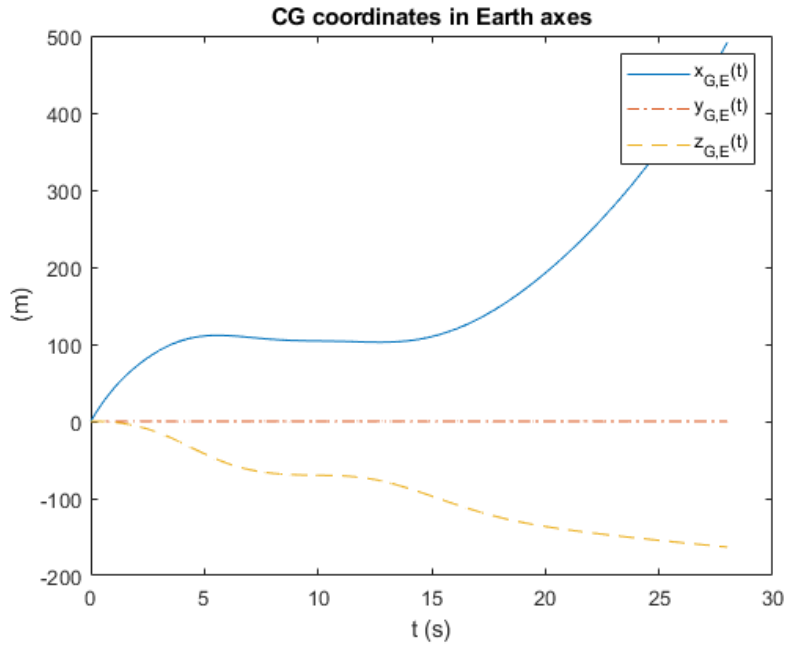

Figura 1.1 Center of gravity position, cobra

```
81      transpose(angle2dcm(fPsi(t),fTheta(t),fPhi(t),'ZYX'))*[u(t);v(t);w(t)
        ]; %  + 0.*Pos;
82
83  %% Solution of navigation equations
84  options = odeset( ...
85      'RelTol', 1e-3, ...
86      'AbsTol', 1e-3*ones(3,1) ...
87      );
88  PosE0 = [0;0;0];
89  [vTime2, vPosE] = ode45(dPosEdt, vTime, PosE0, options);
90  N = length(vPosE);
91
92  vXe = vPosE(:,1); vYe = vPosE(:,2); vZe = Z0 + vPosE(:,3);
```

We note the discontinuity of the angles Θ, Ψ, Φ when Θ tends to $\pi/2$. This behavior is entirely consistent with what has been said about Euler angles and their definition. The quantities were constructed using *function handles*, i.e., anonymous functions that are used only in the code without the need to save them.

1.3.2 Looping Kinematics

Looping is a maneuver performed by pitching up the aircraft so that it executes a complete circular turn (loop). A perfect loop should trace a perfect circular trajectory that ends in exactly the same condition (in terms of position as well as aircraft orientation) in which it began, and during which the angle between the aircraft's linear velocity vector V (tangent to the trajectory) and the nose, or rather the x_B axis of the body frame, remains constant and equal to the initial value. It is a maneuver during which, theoretically, the pilot intervenes only with longitudinal controls, and the aircraft's evolution is contained within a vertical plane. The roll and yaw angular velocities are null throughout the maneuver, as is the component of linear velocity in the direction transverse to the trajectory plane. In a more realistic flight condition, let's assume the pilot is initially

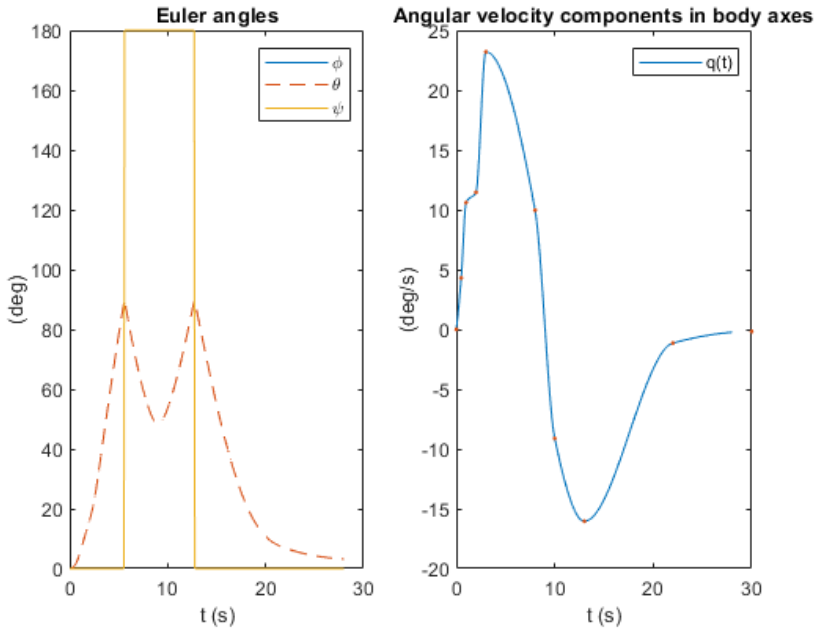

Figura 1.2 Euler angles and angular velocity, cobra

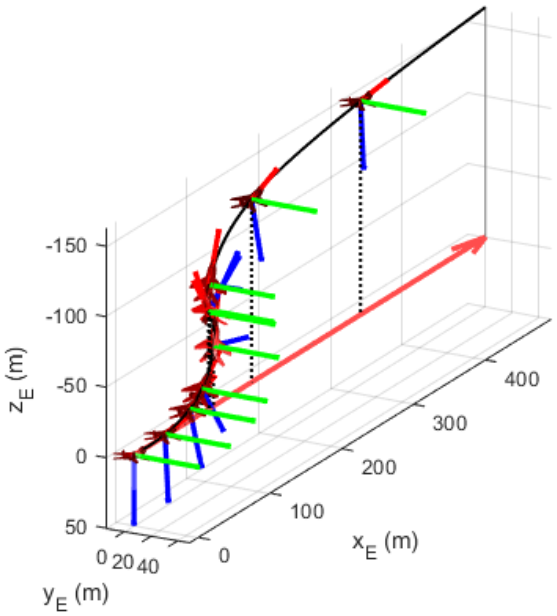

Figura 1.3 Cobra trajectory

flying with level wings, constant speed, and altitude. Then, they pitch up to perform a complete loop by linearly increasing the pitch angular velocity. In the first phase of the looping, it is necessary to apply throttle to compensate for the loss of speed, which is why q(t) reaches q_{max} linearly. In the second phase, the aircraft returns to approximately its initial altitude, continuing in level flight, assuming that in the final phase of the maneuver, q(t) decreases, still linearly, from q_{max} to 0. The quantities were constructed using *function handles*, i.e., anonymous functions used only in the code without needing to be saved, which are widely used both in kinematics examples and in subsequent applications. The time histories of the quaternion components, Euler angles, and the trajectory are reported below.

Listing 1.2

```
%% condizioni iniziali
phi0=0;
psi0=0;
theta0=0;
X0=0;
Y0=0;
Z0=0;
p_max=0;%rad/s
q_max=1;%rad/s
r_max=0;%rad/s
t_fin=5/2*pi;
N=100;
t_span=linspace(0,t_fin,N);
%% Leggi temporali delle componenti della velocita ' angolare del
   velivolo nel riferimento body
p=@(t) 0;

vBreakPointsQ(1,:)=[0, t_fin/5, t_fin*4/5, t_fin]; %vettore dei tempi di
   stop
vBreakPointsQ(2,:)=[0, q_max, q_max,0]; %vettore del valore della
   velocit  q

q=@(t)...
interp1(vBreakPointsQ(1 ,:) ,vBreakPointsQ(2 ,:) ,t);
q_plot=q(linspace(0,t_fin,N));
r=@(t) 0;

%% Leggi temporali delle componenti delle velocit  di traslazione nel
   riferimento body
u0=100;%m/s

vBreakPointsU(1,:)=[0, t_fin/5, t_fin*4/5, t_fin ]; %vettore dei tempi di
    stop
vBreakPointsU(2 ,:)=[u0+10, u0, u0,u0];%vettore del valore della
   velocit  u


u=@(t)...
interp1(vBreakPointsU(1 ,:) ,vBreakPointsU(2 ,:) ,t);

v0=0;%m/s
w0 = 0 ; %m/s

v=@(t) 0;
w=@(t) 0;
%% risoluzione della Gimbal Equation
dQuatdt=@(t ,Q)...
0.5 * [ 0 -p(t) -q(t) -r(t) ;
p(t) 0 r(t) q(t) ;
q(t) -r(t) 0 p(t);
r(t) q(t) -p(t) 0]*Q;

options=odeset ('RelTol' ,1e-9,'AbsTol' ,1e-9*ones (1,4) ) ;
```

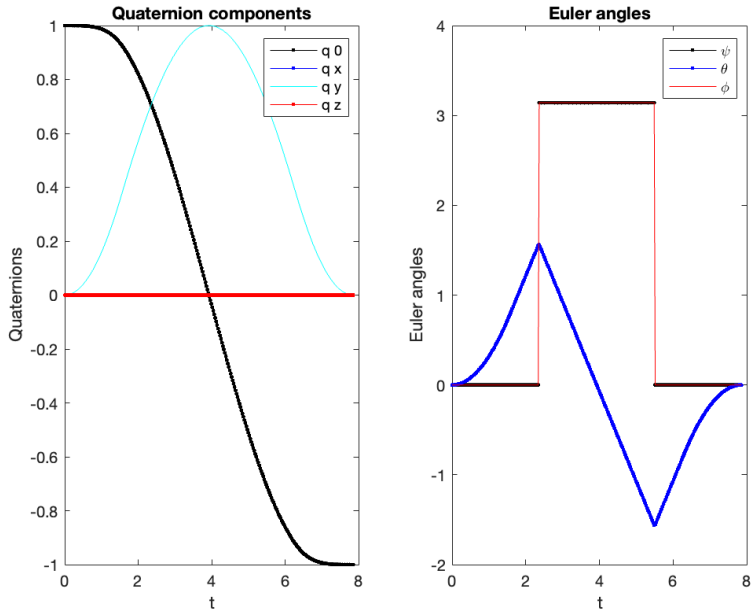

Figura 1.4 Real Looping

```
Q0=angle2quat ( psi0 , theta0 , phi0 ) ;
[vTime,vQuat]=ode45(dQuatdt,[0 t_fin],Q0,options);

%% Grafica per l 'orientamento
quat0 = vQuat(:,1);
quatx = vQuat(:,2);
quaty = vQuat(:,3);
quatz = vQuat(:,4);
figure (1) ;
subplot 121
plot (vTime , quat0 , 'k.-' , vTime , quatx , 'b.-' , vTime , quaty , 'c-
    ',vTime, quatz,'r.-');
legend('q 0', 'q x', 'q y','q z'); title('Quaternion components');
xlabel ('t') ; ylabel ('Quaternions') ; [ vpsi , vtheta , vphi ] =
   quat2angle (vQuat) ;
subplot 122
plot(vTime, vpsi,'k.-', vTime, vtheta, 'b.-', vTime, vphi,'r-');
legend('\psi', '\theta', '\phi');
title('Euler angles'); xlabel('t'); ylabel('Euler angles');
%% risoluzione della Navigation Equation
%Funzione interpolante per conoscere la storia del quaternione
Quat=@(t)...
[interp1( vTime , vQuat (:,1),t),...
interp1(vTime,vQuat(:,2),t) ,...
interp1(vTime,vQuat(:,3),t) ,...
interp1(vTime,vQuat(:,4),t) ];

%Matrice per passare dagli assi terra agli assi body
T_BE=@(Q)...
    quat2dcm (Q) ;%quat2dcm calcola la matrice dei coseni direttori data
   una matrice
%di quaternioni
```

We note in the real looping the discontinuity of the angles Θ, Ψ, Φ when Θ tends to $\pi/2$.

This behavior is entirely consistent with what has been said about Euler angles and their definition; in the inverse matrix of the Gimbal Equation (1.6), there is a $cos(\Theta)$ in the denominator, so for values of Θ close to $\pi/2$, some elements of the transformation matrix tend to infinity, which makes the matrix singular and generates the evident discontinuities in the formulation in terms of Euler angles. As already mentioned, the formulation in terms of the orientation quaternion constitutes a singularity-free parameterization of the attitude; indeed, we do not notice discontinuities in the graph of the quaternion components.
The aircraft's trajectory is reported below.

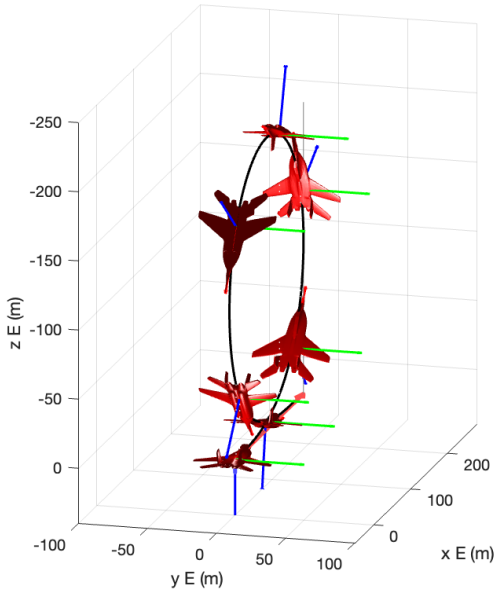

Figura 1.5 Real Trajectory

Capitolo 2

Non linear Aircraft Performance Equations (NAPE)

2.1 Introduction

The study of unsteady aircraft motion is conducted through the analysis of the laws of variation of a number of independent parameters equal to the number of degrees of freedom. The general equations of motion are written starting from the Cardinal Equations of Dynamics:

$$\int_{\mathcal{B}} \frac{dP}{dt} dm = \mathbf{F} \quad \int_{\mathcal{B}} (P - C) \times \frac{d^2P}{dt^2} dm = \mathbf{M} \tag{2.1}$$

which can be expressed in the form:

$$\frac{dQ}{dt} = \mathbf{F} \quad \frac{d\mathcal{K}_r}{dt} + W/g(G - C) \times \frac{d^2C}{dt^2} = \mathbf{M} \tag{2.2}$$

The purpose of this simulation is the analysis of aircraft performance from a translational point of view only. Therefore, the 3 rotational degrees of freedom will not be analyzed, but only the 3 translational ones. The dynamics of this type of motion is governed by the first law of dynamics, which I can rewrite in the form:

$$\frac{W}{g}\frac{d\boldsymbol{V}_G}{dt} = \boldsymbol{F} \tag{2.3}$$

Using Poisson's theorem:

$$\frac{d\boldsymbol{V}_G}{dt} = \dot{\boldsymbol{V}}_{\boldsymbol{G}} + \Omega_m \wedge \boldsymbol{V}_{\boldsymbol{G}} \tag{2.4}$$

The equation becomes, in compact form:

$$\frac{W}{g}\left(\left\{\dot{V}_G\right\}_m + \left[\tilde{\Omega}\right]_m\right]\left[V_G\right]_m\right) = \{F\}_m \tag{2.5}$$

Let's see how this equation comes into play in the study of aircraft navigation, also subject to control laws for achieving or maintaining a certain flight condition.

2.2 Problem Setup

In our discussion, we will make simplifying assumptions and derive interesting insights regarding performance and the control of flight parameters and conditions. Assume the aerodynamic axes system as the reference frame with respect to which the components of force and velocity of the aircraft are referred, projecting the equations of motion. This choice is particularly convenient under the following assumptions about the motion:

- $\beta = 0$ constantly;
- T aligned instant by instant with V
- small incidences

Under these assumptions, we can consider the aerodynamic axes frame to be coincident with the body frame, and the thrust vector will also be aligned with the aerodynamic axes. Projecting the equations governing the dynamics of a rigid body's motion onto the aerodynamic axes, we arrive at the following differential system:

$$\begin{cases} \dot{V} = \frac{T-D}{m} - g\sin(\gamma) \\ \dot{\gamma} = \frac{L\cos(\nu)-mg\cos(\gamma)}{mV} \\ \dot{\delta} = \frac{L\sin(\nu)}{mV\cos(\gamma)} \end{cases} \tag{2.6}$$

These equations are known as *Nonlinear Aircraft Performance Equations* and govern the translational dynamics of an aircraft under the aforementioned assumptions. The first equation expresses the instantaneous linear acceleration of the aircraft as a function of the excess thrust over Drag and the "braking" action due to weight in the direction of the velocity vector. The second equation expresses the temporal variation of the flight path angle as a function of the magnitude of the lift (its vertical component) and its excess with respect to weight. The third equation, finally, expresses the temporal variation of the heading angle, strongly dependent on the bank angle. We can also allow for a variable mass with fuel consumption, indeed we can write:

$$\dot{m} = \frac{-\dot{W}_{fuel}}{g} = -K_{\dot{W}}T \tag{2.7}$$

K_W is the Thrust Specific Fuel Consumption (TSFC) and generally depends on Mach and altitude; however, the function K_W (M, h) is a very sensitive engine datum and difficult to find, so we will assume it to be constant with good approximation. To the NAPE, we must couple the auxiliary kinematic relations, which under the appropriate assumptions in aerodynamic axes are:

$$\begin{cases} \dot{x}_{E,G} = V\cos(\gamma)\cos(\delta) \\ \dot{y}_{E,G} = V\cos(\gamma)\sin(\delta) \\ \dot{h} = V\sin(\gamma) \end{cases} \tag{2.8}$$

Drag can be considered a function of lift through the parabolic polar approximation, an entirely acceptable hypothesis given the small angles of attack we are considering. Equations (2.6),(2.7),(2.8) together with the model for calculating Drag constitute a system of evolution equations expressible in the form:

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) \tag{2.9}$$

where:

- $\boldsymbol{x}$=$[V, \gamma, \delta, m, x_{E,G}, y_{E,G}, h]^T$ is the *state vector and*
- $\boldsymbol{u}$=$[T, L, \nu]^T$ is the vector of input or control variables.

Ultimately, assuming the dynamic model with inputs $\boldsymbol{u}$ and unknowns $\boldsymbol{x}$ is valid, starting from a certain $\boldsymbol{x_0}$, I want to track a new flight condition. For example, let's suppose that from time -∞ to time 0 the aircraft is at a certain altitude at a certain speed, then for t > 0 I want the aircraft to change flight condition. We ask ourselves what are the values of L, ν, and T necessary to do this. During navigation, the instantaneous values of the *commanded thrust* Tc(t), *commanded lift* Lc(t), and *commanded bank angle* $\nu(t)$ will be adjusted. We want to pursue a *commanded velocity* Vc(t), *commanded climb angle* γ_c(t) (therefore a *commanded rate of climb* $V_c \sin(\gamma)_c(t)$ and a *commanded heading angle* δ_c(t). Let's assume the following models for errors and transfer functions are valid:

- $\dot{T}$=$p_T(T_c - T)$ T(s)=$\frac{p_T}{s+p_T}$
- $\dot{L}$=$p_L(L_c - L)$ T(s)=$\frac{p_L}{s+p_L}$
- $\dot{\nu}$=$p_\nu(\nu_c - \nu)$ T(s)=$\frac{p_\nu}{s+p_\nu}$

where p_t, p_L, p_ν are time constants characteristic of the engine and operating conditions (but which we assume constant). The laws that control navigation are precisely the functions L_c(t),T_c(t),ν_c(t). The first two are obtained with a *proportional-integral* (PI) control logic, where the commanded variable is proportional to the error and its integral; the third is obtained with a *proportional* control logic, where ν_c is proportional to the error. In the 3 cases, the errors are defined as:

- $E_V(t)$=$V_c - V(t)$
- $E_h(t)$=$V_c[\sin(\gamma_c) - \sin(\gamma)(t)]$
- $E_\delta(t)$=$\delta_c - \delta(t)$

2.3 Lockheed C-130 Mission Simulation

A possible extension of the application just seen concerns establishing a sequence of several successive commanded values of velocity, climb angle, and turn. In this example, 4 possible flight conditions of a Lockheed C-130 will be analyzed, whose main characteristics are summarized in the following table.

Quantità	Valore	Unità
Superficie alare	3010	ft^2
Apertura alare	146	ft
Fattore di Oswald	0.95	-
Coefficiente di resistenza parassita	0.02	-
Peso massimo	327000	lbf
Peso a vuoto	157000	lbf
Peso a serbatoi pieni	170000	lbf
Consumo specifico	4e-6	$\frac{slug}{lbf*s}$
Velocità minima e massima	200-600	mph
Spinta massima	72000	lbf
Massimo angolo di bank	30	deg
Velocità iniziale	450	mph
x_{E_G}, y_{E_G}	[0 0]	ft
Quota iniziale	20000	ft
Angolo di bank iniziale	0	deg
Angolo di heading iniziale	0	deg
$C_{L\alpha}$	0.1	$\frac{1}{deg}$
α_{0L}	-2.86	deg
Vettore vento iniziale	[25,25,0]	mph
Angoli di salita, heading e bank iniziali	[0,0,0]	deg
p_T, p_L, p_ν	[2, 2.5,1]	$\frac{1}{s}$
k_{TI}, k_{TP}, k_{LI}, k_{LP}, k_ν	[2e-3, 8e-2, 1e-2, 5e-1, 7.5e-2]	s
Velocità comandata(1)	450	mph
Angolo di salita comandato(1)	5	deg
Heading comandato(1)	15	deg
Velocità comandata(2)	420	mph
Angolo di salita comandato(2)	0	deg
Heading comandato(2)	15	deg
Velocità comandata(3)	400	mph
Angolo di salita comandato(3)	-5	deg
Heading comandato(3)	0	deg
Velocità comandata(4)	400	mph
Angolo di salita comandato(4)	0	deg
Heading comandato(4)	0	deg

Figura 2.1 Lockheed C-130

It is necessary to make some modifications to the block that characterizes the desired operating condition of the aircraft. In particular, the use of *if* statements was chosen starting from the activation condition of the scheduling, which occurs at t = 5s, using two blocks present in Simulink. The *Clock* block contains the current simulation time; its use is fundamental in this application given the change in commanded quantities at a certain simulation time. The *Switch* block is the Simulink equivalent of if-else statements. It has 3 input ports where the second is the condition that must be met for the if to be valid. The if corresponds to the first port, the $else$ to the third. Ultimately, the output will be the input of port 1 if the condition is met, and port 3 if it is not.
The code will be the same as for a single condition of commanded variables, but the *Desired* A/C *State* block will be different and is as follows, with:

- input of commanded variables(1) starting from 5 seconds.
- input of commanded variables(2) starting from 95 seconds.
- input of commanded variables(3) starting from 155 seconds.
- input of commanded variables(4) starting from 245 seconds.

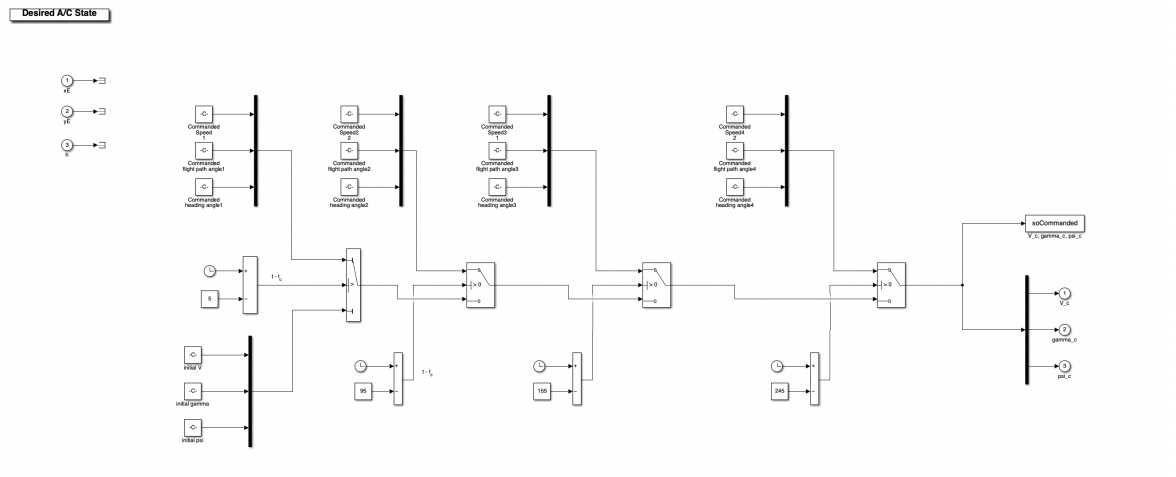

Figura 2.2 *Desired A/C State* Block

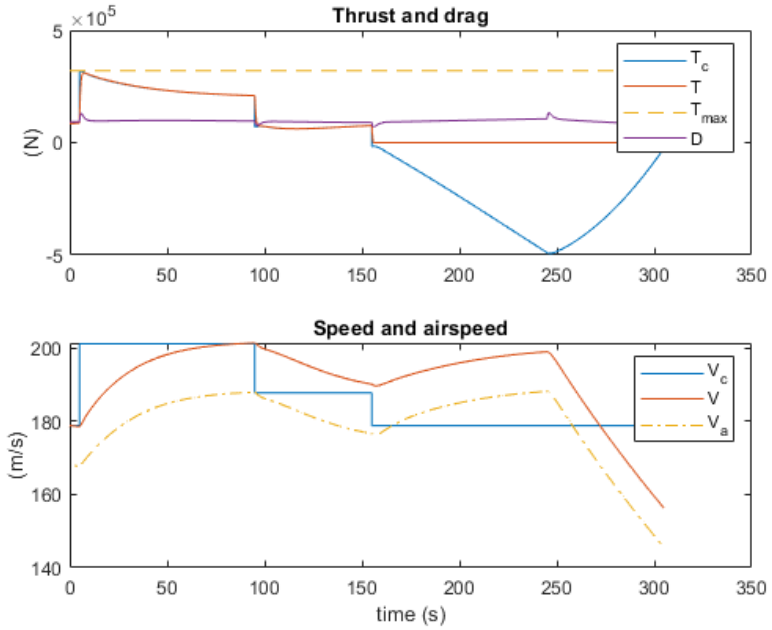

Figura 2.3 Results of velocity and Thrust in the climb and right turn maneuver

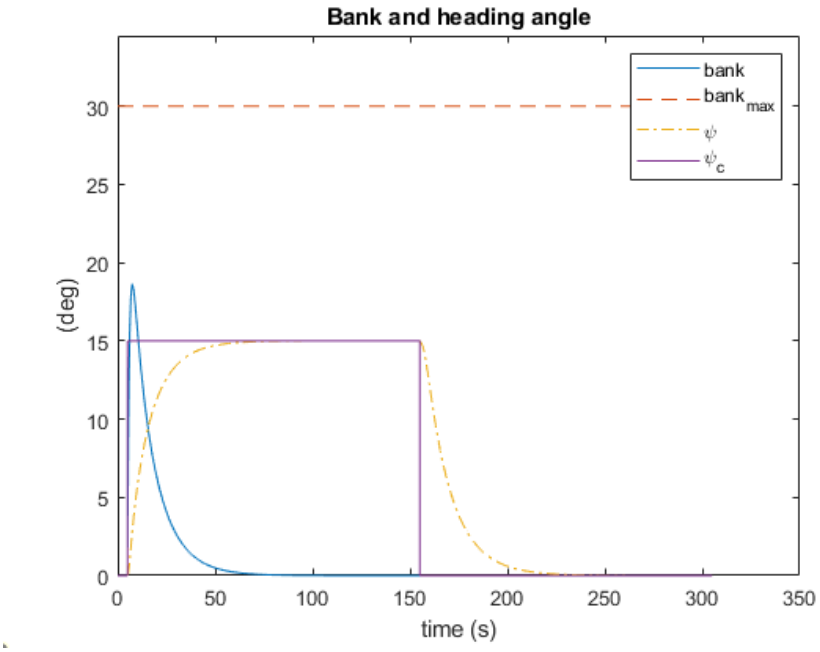

Figura 2.4 Results of the wing bank angle and heading angle

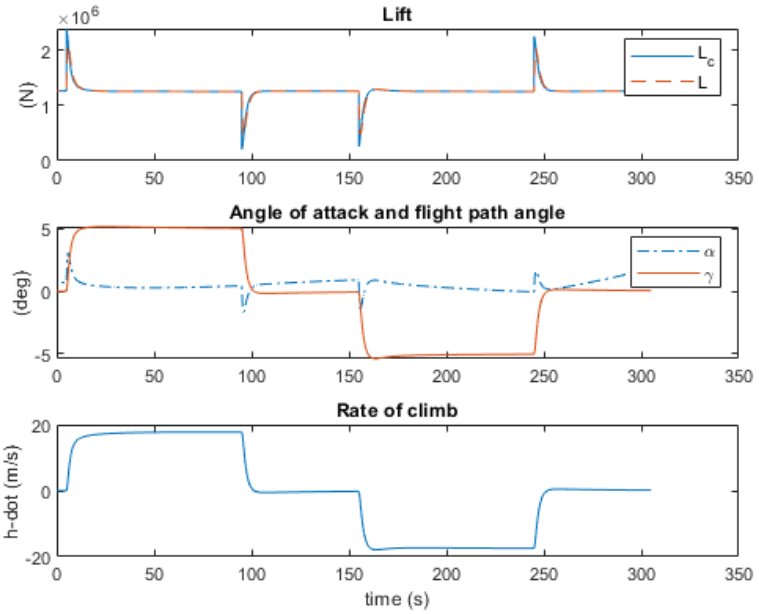

Figura 2.5 Results for lift, angle of attack, and climb rate

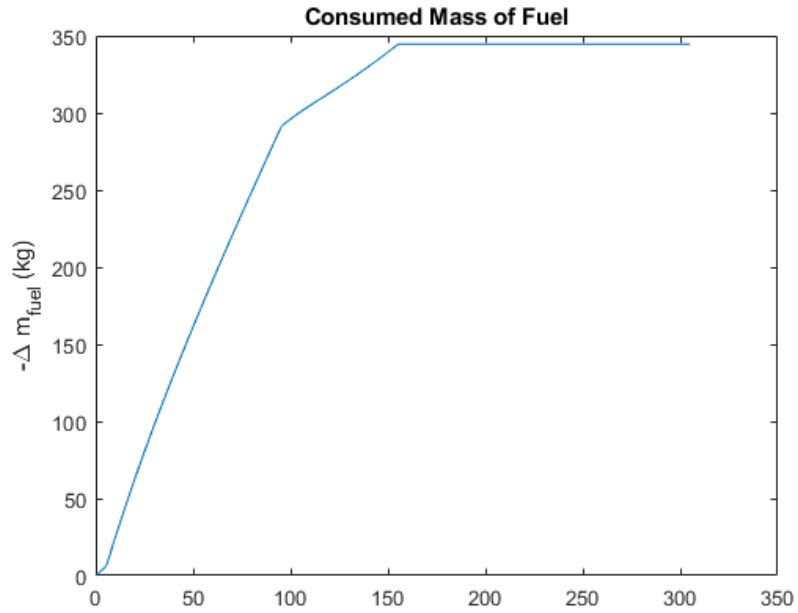

Figura 2.6 Result of fuel mass consumption expressed in kg

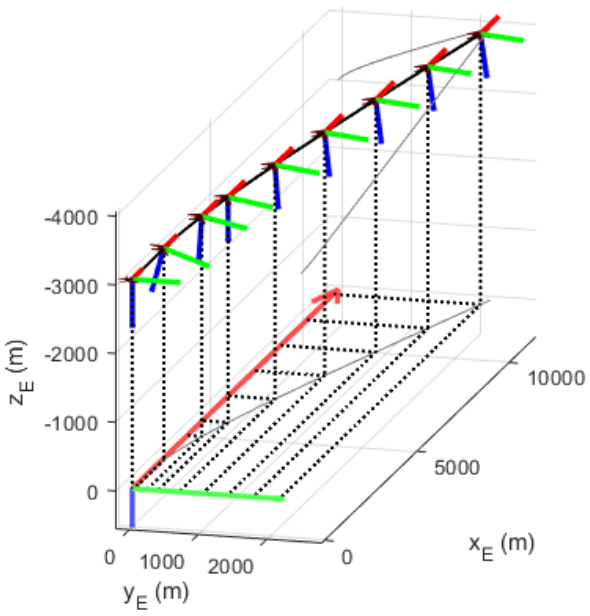

Figura 2.7 Trajectory described by the aircraft

Capitolo 3

Equations of Motion

3.1 Introduction

This notebook deals with projecting the equations of motion of an aircraft with 6 Degrees of Freedom (DoF) onto an appropriate reference axes system and developing a suitable model for aerodynamic and propulsive forces. These dynamic equations, coupled with auxiliary kinematic equations, result in a system of 12 (or 13) equations in as many unknowns which, given an initial condition, can be integrated to obtain the time histories of the flight. The cardinal equations of dynamics, expressed in an aircraft reference frame, under the assumption of a rigid aircraft, are written as:

$$\begin{Bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{Bmatrix} = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} + \frac{g}{W} \begin{Bmatrix} X_G + X_A + X_T \\ Y_G + Y_A + Y_T \\ Z_G + Z_A + Z_T \end{Bmatrix} \tag{3.1}$$

$$\begin{Bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{Bmatrix} = \frac{1}{det[I]_B} \begin{bmatrix} I_1 & I_2 & I_3 \\ I_2 & I_4 & I_5 \\ I_3 & I_5 & I_6 \end{bmatrix} \left(\begin{Bmatrix} L_A + L_T \\ M_A + M_T \\ N_A + N_T \end{Bmatrix} - \begin{bmatrix} 0 & -p & -r \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} I_{xx} & -I_{yx} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \right) \tag{3.2}$$

It should be noted that this system of 6 equations is not sufficient to determine the motion as they are equations obtained by projecting the laws of dynamics onto a frame whose position and orientation are themselves unknowns. It should also be observed that the terms on the right-hand side of the equations also depend on position and attitude, which is why we need to couple these equations with the kinematic equations to solve the motion. To convince ourselves of this, it is enough to observe that decomposing the weight force along the 3 axes involves Euler angles, which are unknowns. To solve these equations, it is necessary to implement a constitutive model for aerodynamics and propulsion: that is, to assume analytical formulations that allow us to evaluate aerodynamic forces and moments as a function of state variables, control variables, and operating conditions. In a completely general 6-DoF treatment, the state vector is composed of 12 elements, 6 kinematic unknowns that make up the vector $\mathbf{x_k}$ and 6 dynamic

unknowns that make up the vector $\mathbf{x_d}$, where:

- $\mathbf{x_k}$=$[x_{EG}, y_{EG}, z_{EG}, \Phi, \Theta, \Psi]^T$
- $\boldsymbol{x_d}$=$[u, v, w, p, q, r]^T$

It is often preferable to express the translational dynamics equations (3.1) of an aircraft in terms of $\dot{V}$, $\dot{\alpha}$, $\dot{\beta}$, rather than in terms of u, v, w, as it is common for the dimensionless coefficients of aerodynamic forces and moments that appear to be obtained through experimental tests and through functional dependencies on the same aerodynamic angles.

Note first of all that it is possible to switch from one frame to another through the relations:

$$V^2 = u^2 + v^2 + w^2 \tag{3.3}$$

$$\alpha = atan(w/u) \tag{3.4}$$

$$\beta = atan(v/V) \tag{3.5}$$

$$u = Vcos(\alpha)cos(\beta) \tag{3.6}$$

$$v = Vsin(\beta) \tag{3.7}$$

$$w = Vcos(\alpha)sin(\beta) \tag{3.8}$$

Substituting into (3.1) yields the following equations:

$$\begin{cases} \dot{V} = \frac{1}{m}\begin{bmatrix} -D\cos\beta + Y_A\sin\beta + X_T\cos\alpha\cos\beta + Y_T\sin\beta + \\ +Z_T\sin\alpha\cos\beta - mg(\cos\alpha\cos\beta\sin\theta - \sin\beta\sin\phi\cos\theta - \sin\alpha\cos\beta\cos\phi\cos\theta) \end{bmatrix} \\ \dot{\alpha} = \frac{1}{mV\cos\beta}\begin{bmatrix} -L + Z_T\cos\alpha - X_T\sin\alpha + mg(\cos\alpha\cos\phi\cos\theta + \\ +\sin\alpha\sin\theta) \end{bmatrix} + q - \tan\beta(p\cos\alpha + r\sin\alpha \\ \dot{\beta} = \frac{1}{mV}\begin{bmatrix} D\sin\beta + Y_A\cos\beta - X_T\cos\alpha\sin\beta + Y_T\cos\beta - Z_T\sin\alpha\sin\beta + \\ +mg(\cos\alpha\sin\beta\sin\theta + \sin\phi\cos\beta\cos\theta - \sin\alpha\sin\beta\cos\phi\cos\theta) \end{bmatrix} + p\sin\alpha - r\cos\alpha \end{cases} \tag{3.9}$$

3.2 6-DoF Motion

This chapter studies the motion of the F/A-18 High Angle-of-attack Research Vehicle (HARV), provided by the instructor in the teaching material. This model is non-linear and has been deduced from flight experiments. Functions for calculating aerodynamic coefficients have been implemented in accordance with the data provided by the model. First, initial data are loaded from a previously compiled .mat file. The DSVAircraft class and the myAC object are defined. Appropriate initial conditions are assigned, and control laws are provided to perform a specific maneuver with functions defined on the fly. The maneuver considered is the approach to a landing runway. Subsequently, the components of external forces and moments along the body axes are calculated. A function implementing the complete system of equations is used, and a step-by-step integration

method with the ode45 function is used. The calculation code is presented, and the plotting lines will be omitted.

Listing 3.1

```
global g...                    %Accelerazione di gravit
       myAC                    %Oggetto 'Velivolo'

%  Definizione della classe DSVAircraft e dell'oggetto 'Velivolo'
aircraftDataFileName = 'HARV.txt';
myAC = DSVAircraft(aircraftDataFileName);

%  Costanti e condizioni iniziali
g = 9.81; %Accelerazione di gravit  [m/s^2]
xEG_0 = 0; %[m]
yEG_0 = 0; %[m]
zEG_0 = 0; %Altitudine [m]
[air_Temp0,sound_speed0,air_pressure0,rho0] = atmosisa(-zEG_0);

%%%%% ANGOLI DI EULERO E QUATERNIONE INIZIALI (MIO INTERVENTO)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
phi0 = convang(0.000,'deg','rad'); %Angolo di bank
theta0 = convang(4.460,'deg','rad'); %Angolo di elevazione
psi0 = convang(0.000,'deg','rad'); %Angolo di heading
eul0=[psi0,theta0,phi0]
quat_0=angle2quat(psi0,theta0,psi0);
q1_0=quat_0(1);
q2_0=quat_0(2);
q3_0=quat_0(3);
q4_0=quat_0(4);

% velocit  angolari iniziali
p0 = convangvel(0.000,'deg/s','rad/s'); %Velocit  angolare di rollio
q0 = convangvel(0.000,'deg/s','rad/s'); %Velocit  angolare di beccheggio
r0 = convangvel(0.000,'deg/s','rad/s'); %Velocit  angolare di imbardata
M0 = 0.246010; %Numero di Mach
V0 = M0*sound_speed0; %Velocit  lineare del baricentro
alpha_B0 = convang(4.557754,'deg','rad'); %Angolo di attacco valutato
    rispetto l'asse x Body
beta_der0 = convang(0.000,'deg','rad'); %Angolo di derapata
u0 = V0*cos(beta_der0)*cos(alpha_B0); %Componente della velocit  lineare
     del baricentro lungo l'asse x Body
v0 = V0*sin(beta_der0); %Componente della velocit  lineare del
    baricentro lungo l'asse y Body
w0 = V0*cos(beta_der0)*sin(alpha_B0); %Componente della velocit  lineare
     del baricentro lungo l'asse z Body

%  Comandi di volo iniziali
delta_a0 = convang(0.000,'deg','rad');
delta_e0 = convang(-10.614717,'deg','rad');
delta_r0 = convang(0.000,'deg','rad');
delta_T0 = 0.474918;

%% Integrazione delle equazioni del moto a 6-DoF
t_fin = 230; %Tempo di simulazione [s] (Manovra di avvicinamento alla
    pista di atterraggio di un aeroporto)
```

```
state_0 = [u0,v0,w0,p0,q0,r0,xEG_0,yEG_0,zEG_0,q1_0,q2_0,q3_0,q4_0]; %
    %%%%%%%%%%%%(MIO INTERVENTO)%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%  Assegnazione delle leggi temporali dei comandi di volo
global delta_a delta_e delta_r delta_T

%Possibile manovra di avvicinamento alla pista di atterraggio di un
    aeroporto
delta_a_exc1 = convang(-5,'deg','rad');
delta_a = @(t) interp1([0, 40, 41, 45, 46, t_fin],...
                       [delta_a0, delta_a0, delta_a_exc1, delta_a_exc1,
    delta_a0, delta_a0],t,'linear');

delta_e_exc1 = convang(-9.5,'deg','rad');
delta_e_exc2 = convang(-10,'deg','rad');
delta_e = @(t) interp1([0, 20, 80, 180, t_fin],...
                       [delta_e0, delta_e0, delta_e_exc1, delta_e_exc2,
    delta_e_exc2],t,'linear');

delta_r_exc1 = convang(0,'deg','rad');
delta_r = @(t) interp1([0, 200, 203, 205, 208, t_fin],...
                       [delta_r0, delta_r0, delta_r_exc1, delta_r_exc1,
    delta_r0, delta_r0],t,'linear');

delta_T_exc1 = delta_T0 ;
delta_T = @(t) interp1([0, t_fin],[delta_T0 delta_T_exc1],t,'linear');

%  Integrazione del sistema di equazioni differenziali
options = odeset('RelTol',1e-9,'AbsTol',1e-9*ones(1,13));
[vTime,mState] = ode45(
    @eqLongDynamicStickFixed_6DoF_uvwpqr_EulerQ7n2_PROVOIO,[0 t_fin],
    state_0,options);
```

Listing 3.2

```
global g...
       delta_a delta_e delta_r delta_T...
       myAC

%  Assegnazione delle componenti del vettore di stato
u = state(1);
v = state(2);
w = state(3);
p = state(4);
q = state(5);
r = state(6);
xEG = state(7);
yEG = state(8);
zEG = state(9);
q1 = state(10);
q2 = state(11);
q3 = state(12);
q4 = state(13);

quat=[q1,q2,q3,q4];

```

```
[psi,theta,phi]=quat2angle(quat,"ZYX");

%% Quantit  caratteristiche
%  Equazioni generali del moto del velivolo
omegatilde = [0, -r,  q;
              r,  0, -p;
             -q,  p,  0];

%  Equazioni cinematiche ausiliari
T_BE = quat2dcm(quat);
T_EB = T_BE';
T_gimb_q = 0.5*[-q2,   -q3,   -q4;
                 q1,    -q4,    q3;
                 q4,     q1,   -q2;
                -q3,    q2     q1];

%Parametri cinematici
V = (u^2 + v^2 + w^2)^(1/2);
alpha_B = atan(w/u);
beta_der = asin(v/V);

%Propriet  termodinamiche
rho = density(-zEG);

%% Definizione delle forze e dei momenti agenti sul velivolo
%  Componenti in assi velivolo della forza risultante
L = 0.5*rho*V^2*myAC.S*CL(convang(alpha_B,'rad','deg'),...
                          convang(delta_e(t),'rad','deg'));
D = 0.5*rho*V^2*myAC.S*CD(convang(alpha_B,'rad','deg'));
Y_A = 0.5*rho*V^2*myAC.S*CY_A(convang(alpha_B,'rad','deg'),...
                              convang(beta_der,'rad','deg'),...
                              convang(delta_a(t),'rad','deg'),...
                              convang(delta_r(t),'rad','deg'));
X_T = delta_T(t)*myAC.T_max_SL*cos(myAC.mu_T);
%Come richiesto, si assume che la spinta propulsiva massima disponibile
    sia
%costante al variare della quota e della velocit  e che sia pari al
    valore
%della spinta propulsiva massima disponibile al livello del mare.
Y_T = 0;
Z_T = delta_T(t)*myAC.T_max_SL*sin(myAC.mu_T);
X = X_T - D*cos(alpha_B) + L*sin(alpha_B) - myAC.W*sin(theta);
Y = Y_T + Y_A + myAC.W*sin(phi)*cos(theta);
Z = Z_T - D*sin(alpha_B) - L*cos(alpha_B) + myAC.W*cos(phi)*cos(theta);

%  Componenti in assi velivolo del momento risultante
L_roll_A = 0.5*rho*V^2*myAC.S*myAC.b*Croll(convang(alpha_B,'rad','deg')
    ,...
                                           convang(beta_der,'rad','deg')
    ,...
                                           convang(delta_a(t),'rad','deg'
    ),...
                                           convang(delta_r(t),'rad','deg'
    ),...
                                           p,r);
L_roll_T = 0;
```

```
M_pitch_A = 0.5*rho*V^2*myAC.S*myAC.mac*Cpitch(convang(alpha_B,'rad','deg'),...
                                               convang(delta_e(t),'rad','deg'),...
                                               q);
C_M_T = (delta_T(t)*myAC.T_max_SL*myAC.e_T)/(0.5*rho*V^2*myAC.S*myAC.mac);
M_pitch_T = 0.5*rho*V^2*myAC.S*myAC.mac*C_M_T;
N_yaw_A = 0.5*rho*V^2*myAC.S*myAC.b*Cyaw(convang(alpha_B,'rad','deg'),...
                                         convang(beta_der,'rad','deg'),...
                                         convang(delta_a(t),'rad','deg'),...
                                         convang(delta_r(t),'rad','deg'),...
                                         r);
N_yaw_T = 0;

%  Costruzione della funzione integranda
dStatedt = ([-omegatilde,                          zeros(3);
             zeros(3),       myAC.I_matrix\(-omegatilde*myAC.I_matrix);
              T_EB,                                zeros(3);
             zeros(4,3),                         T_gimb_q]*[u;
                                                            v;
                                                            w;
                                                            p;
                                                            q;
                                                            r])+...
          [(g/myAC.W)*X;
           (g/myAC.W)*Y;
           (g/myAC.W)*Z;
           myAC.I_matrix\[L_roll_A + L_roll_T;
                          M_pitch_A + M_pitch_T;
                          N_yaw_A + N_yaw_T];
           zeros(7,1)];
%dstatedt[13x1] = Matrix[13x6]*Matrix[6x1] + Matrix[13x1]
```

3.3 Longitudinal-Symmetric Motion

Longitudinal-symmetric motion is a special case of 6-DoF flight that occurs under certain assumptions that reduce the aircraft's degrees of freedom to 3 (this is also why it's called 3-DoF motion). We assume the aircraft has a longitudinal plane of symmetry (both geometric and material), that the propulsive action lies in the plane of symmetry, and that the flight occurs with level wings, from which it follows:
$\Phi(t)=0$
The aircraft will constantly maintain its plane of symmetry vertical. It can also be assumed that the heading angle is constantly null without loss of generality:
$\Psi(t)=0$
Other assumptions that stem from these considerations are:
$v(t) = \beta(t) = 0, p(t) = r(t) = 0, Y(t) = L(t) = N(t) = 0$
It is also specified that the controls the pilot can act upon are only the symmetric ones δ_c, δ_g, δ_T so as not to trigger lateral-directional dynamics.

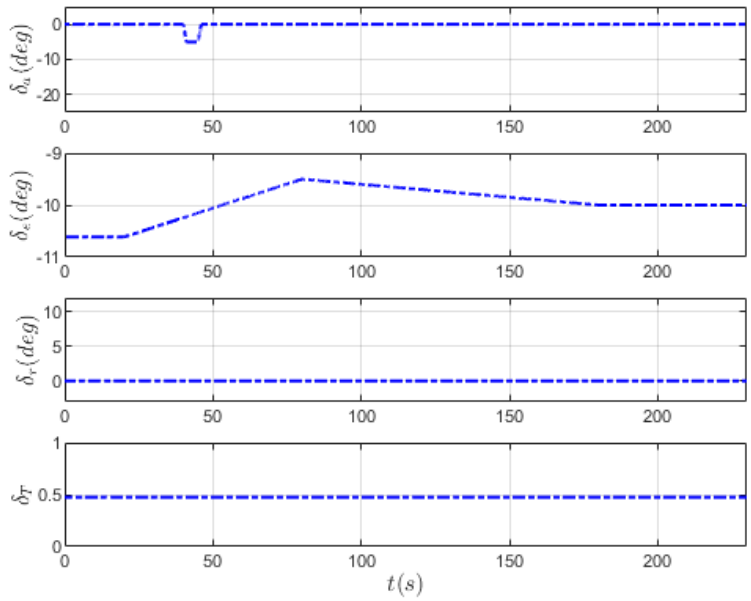

Figura 3.1 Time history of input variables

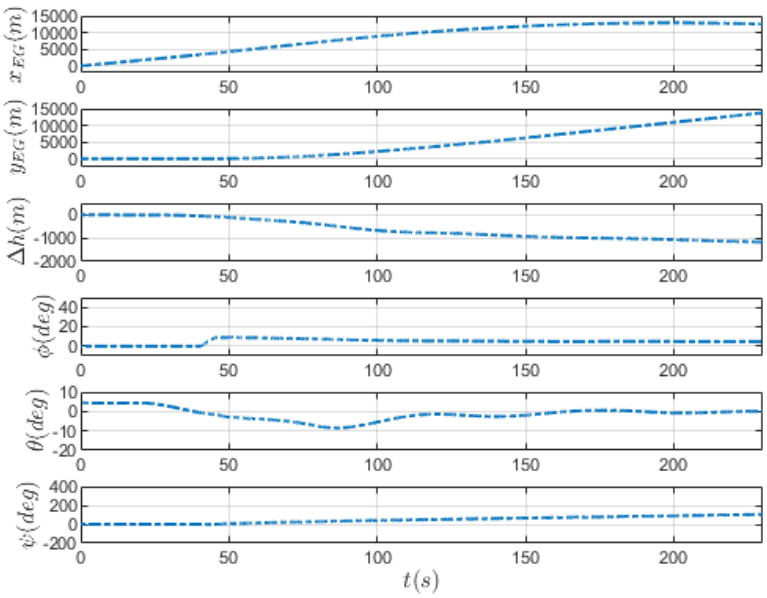

Figura 3.2 Time history of kinematic state variables

By virtue of the assumptions made, we can neglect:

- The second translational equilibrium equation (with unknowns u, v, w).
- The third translational equilibrium equation (with unknowns V, α, β).
- The first and third rotational equilibrium equations.
- The second equation of the navigation equations system.
- The first and third equations of the gimbal equations system.

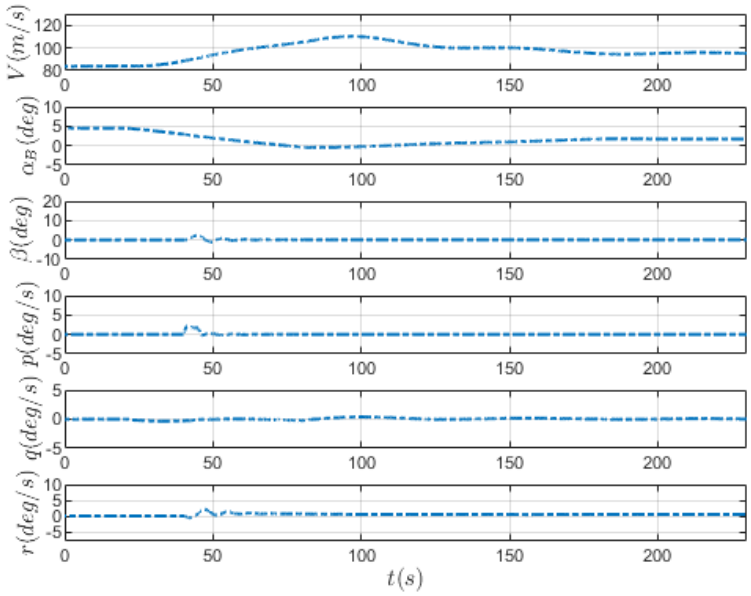

Figura 3.3 Time history of dynamic state variables

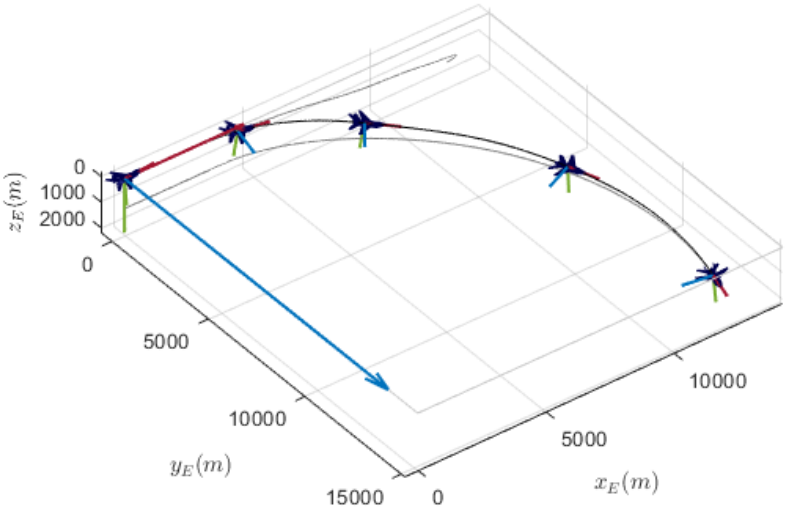

Figura 3.4 Trajectory

The state vector will consist of 3 dynamic unknowns and 3 kinematic unknowns. Choosing V, α as unknowns: $\mathbf{x} = [V, \alpha, q, x_{EG}, y_{EG}, z_{EG}]$. The system of equations characterizing the dynamics of motion simplifies considerably:

$$\begin{cases} \dot{V} = \frac{1}{m}\left[-D + X_T \cos\alpha + Z_T \sin\alpha - mg(\cos\alpha \sin\theta - \sin\alpha \cos\theta\right] \\ \dot{\alpha} = \frac{1}{mV}\left[-L + Z_T \cos\alpha - X_T \sin\alpha + mg(\cos\alpha \cos\theta + \sin\alpha \sin\theta\right] + q \\ I_y \dot{q} = M_A + M_T \end{cases} \tag{3.10}$$

$$\begin{cases} \dot{x_{E,g}} = V(\cos\alpha\cos\theta + (V\sin\alpha\sin\theta) \\ \dot{z_{E,g}} = -V(\cos\alpha\sin\theta + (V\sin\alpha\cos\theta) \\ \dot{\theta} = q \end{cases} \tag{3.11}$$

To have a well-posed problem, it remains to specify the expressions that characterize aerodynamic and propulsive actions. If we assume the parabolic polar model is valid:
$D = \frac{1}{2}\rho V^2 S C_L$ with $C_D = C_{D0} + kC_L^2$
If we hypothesize a linear relationship between the lift coefficient and state and input parameters and model unsteady effects linearly:
$C_L = C_{L\alpha}\alpha + \frac{\bar{c}}{2V}(C_{L\dot{\alpha}}\dot{\alpha} + C_{Lq}q + C_{L\delta_e}\delta_e + C_{L\delta_s}\delta_s$
And, similarly for the pitching moment:
$C_M = C_{M\alpha} + \frac{\bar{c}}{2V}(C_{M\dot{\alpha}}\dot{\alpha} + C_{Mq}q + C_{M\delta_e}\delta_e + C_{M\delta_s}\delta_s$
The definitive system of differential equations can be written as:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & M_{32} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \dot{V} \\ \dot{\alpha} \\ \dot{q} \\ \dot{q} \\ \dot{\theta} \end{Bmatrix} = \begin{Bmatrix} f_1(V,\alpha, \quad ,z_{E,G},\theta,\delta_e,\delta_s) \\ f_2(V,\alpha,q,z_{E,G},\theta,\delta_e,\delta_s) \\ f_3(V,\alpha,q,z_{E,G}, \quad ,\delta_e,\delta_s) \\ f_4(V,\alpha, \quad ,\theta, \quad , \quad) \\ f_5(V,\alpha, \quad ,\theta, \quad , \quad) \\ f_6(\quad , \quad ,q, \quad , \quad , \quad) \end{Bmatrix} \tag{3.12}$$

Where:
M_{32}=$-\frac{c^2 C_{M\dot{\alpha}} V}{4\nu k_{\dot{y}}^2 b}$
Therefore, we can rewrite the equations of longitudinal-symmetric motion in explicit form with respect to the derivative of the state vector by inverting the matrix as:

$$\begin{Bmatrix} \dot{V} \\ \dot{\alpha} \\ \dot{q} \\ x_{E,G} \\ \dot{y_{E,G}} \\ \dot{\theta} \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ -M_{32}f_2 + f_3 \\ f_4 \\ f_5 \\ f_6 \end{Bmatrix} \tag{3.13}$$

The equations are strongly non-linear, which is why it is necessary to integrate them numerically. The problem admits a solution by assigning an appropriate set of initial conditions, which requires solving a problem related to them. In particular, supposing we want to fix an initial condition of balanced flight at a certain altitude, a certain speed, and with fixed γ and q. The initial conditions are therefore not all specified; we do not completely know the vectors $\mathbf{x}(\mathbf{0})$ and $\mathbf{u}(\mathbf{0})$. It remains to determine $\alpha(0)$, $\delta_e(0)$, $\delta_s(0)$, $\delta_T(0)$. The complete set of trimmed flight parameters can be determined by considering that for the flight condition to be balanced, it must necessarily result that $\dot{V} = \dot{\alpha} = \dot{q} = 0$, i.e., the derivatives of the dynamic unknowns must be null. (3.13):

$$\begin{cases} f_1 = 0 \\ f_2 = 0 \\ f_3 = 0 \end{cases} \tag{3.14}$$

It should be noted that the problem of determining trim conditions can admit either 0, 1, or ∞^1 solutions. If we did not fix one of the 4 variables, the system would admit ∞^1 solutions; this is completely in line with the physics of the problem, as if I have more controls, I have more ways to reach a certain desired flight condition. If speed and altitude were outside the flight envelope, the solution would diverge and would not have a real correspondence with the physics of the problem. With an adequate assignment of these two conditions and by fixing one of the 4 unknowns, the problem is well-posed. An *iterative method* can be implemented to find the zero of the *cost function* or *objective function* J, defined as:

$$J(V_0, \alpha(0), z_{E,G}(0), \gamma(0) + \alpha(0), \delta_e(0), \delta_s(0), \delta_T(0)) = f_1^2 + f_2^2 + f_3^2 \tag{3.15}$$

We observe that the function J is non-negative, so seeking the minimum of this function corresponds to seeking precisely the trim condition specified in the system of equations (3.14). The use of the *fmincon* function is fundamental for solving this problem, as it finds a constrained minimum of multivariable functions, attempting to solve problems of the form: min F(X) subject to:

- $AX \leq B$, AeqX = Beq (linear constraints)
- $C(X) \leq 0$, Ceq(X) = 0 (non-linear constraints)
- $LB \leq X \leq UB$(bounds)

3.4 Search for trim conditions with fixed δ_s varying V_0,h_0,γ_0,q_0

This section proposes the solution to the problem detailed in the previous lines, i.e., finding the initial *trimmed* condition by assigning h and V. In this code, it was necessary to fix δ_s because the problem also includes δ_t among the unknowns; we initially fix δ_s = 1. It is possible to appropriately modify the *fmincon* function to constrain the value of the variable δ_s. In this case, the vector ξ will have its third component fixed, so it is possible to construct a system that satisfies the following linear equations:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \alpha \\ \delta_\mathrm{e} \\ \delta_\mathrm{s} \\ \delta_\mathrm{T} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ \delta_{\mathrm{s},0} \\ 0 \end{Bmatrix} \tag{3.16}$$

The equality relation (3.16), in which the matrix Aeq on the left-hand side and the column vector beq on the right-hand side are recognized, is a constraint equation that is equivalent to fixing the value of $\xi_3 = \delta_{s0}$. The output will be the trim condition. The code for this problem is reported below:

Listing 3.3

```
clear all; close all; clc;

disp('Moto del velivolo a 3 gradi di libert ');
disp('Risoluzione del problema di trim ad una data altitudine e velocit
    di volo');
```

```
5
6  %% Dichiarazione delle variabili globali
7  global g...                    %Accelerazione di gravit
8         zEG_0 V0 q0 gamma0...   %Condizioni iniziali
9         rho0 ...                %Densit  dell'aria all'altitudine h = (-
       zEG_0)
10        myAC                    %Oggetto 'Velivolo'
11
12 %% Ricerca delle condizioni di trim
13 aircraftDataFileName = 'DSV_Aircraft_data.txt';
14
15 %  Definizione dell'oggetto 'Velivolo'
16 myAC = DSVAircraft(aircraftDataFileName);
17 %L'oggetto 'myAC'    un'istanza della classe DSVAircraft.
18 %Tutte le variabili membro di 'myAC' risultano essere accessibili
19 %e modificabili dall'utente.
20
21 if (myAC.err == -1)
22     disp('Terminazione.')
23 else
24     disp(['File ',aircraftDataFileName,' letto correttamente.']);
25
26     % Costanti e condizioni iniziali
27     g = 9.81; %Accelerazione di gravit  [m/s^2]
28     xEG_0 = 0; %[m]
29     zEG_0 = -4000; %Altitudine [m]
30     Nvel=20;
31     vV0 = linspace(220.0,350.0,Nvel); %Velocit  di volo [m/s]
32     q0 = convangvel(0.000,'deg/s','rad/s'); %Velocit  angolare di
       beccheggio
33     gamma0 = convang(0.000,'deg','rad'); %Angolo di rampa
34     [air_Temp0,sound_speed0,air_pressure0,rho0] = atmosisa(-zEG_0);
35
36     %% Processo di minimizzazione della funzione di costo
37     %  Valore di tentativo iniziale per il design vector
38     x0 = [0;    %Valore di tentativo iniziale per alpha_0 espresso in rad
39           0;    %Valore di tentativo iniziale per delta_e_0 espresso in
       rad
40           0;    %Valore di tentativo iniziale per delta_s_0 espresso in
       rad
41           0.5]; %Valore di tentativo iniziale per delta_T_0
42     %Il valore di tentativo iniziale viene dedotto considerando una
       condizione
43     %media tra i limiti inferiore e superiore.
44
45     %  Limiti
46     lb = [convang(-15,'deg','rad'),... %Valore minimo per alpha
47           convang(-20,'deg','rad'),... %Valore minimo per delta_e0
48           convang(-5,'deg','rad'),...  %Valore minimo per delta_s0
49           0.2];                        %Valore minimo per delta_T0
50     ub = [convang(15,'deg','rad'),...  %Valore massimo per alpha
51           convang(13,'deg','rad'),...  %Valore massimo per delta_e0
52           convang(5,'deg','rad'),...   %Valore massimo per delta_s0
53           1.0];                        %Valore massimo per delta_T0
54
55     %  0pzioni di ricerca del minimo
```

```
    options = optimset('tolfun',1e-9,'Algorithm','interior-point');

    %  Chiamata alla funzione 'fmincon'
valpha0_deg=zeros(Nvel,1);
vdelta_e0_deg=zeros(Nvel,1);
vdelta_T0=zeros(Nvel,1);
vdelta_s0_deg=zeros(Nvel,1);

% constrain lineari
delta_s0_rad=convang(1,'deg','rad');
Aeq=zeros(4);
Aeq(3,3)=1;
Beq=zeros(4,1);
Beq(3,1)=delta_s0_rad;

for i=1:Nvel

    V0=vV0(i);
    [x,fval] = fmincon(@costLongEquilibriumStaticStickFixed,... %Funzione
    obiettivo del processo di minimizzazione
                       x0,...
                       [],[],Aeq,Beq,... %Vincoli lineari
                       lb,ub,...
                       @myNonLinearConstraint,... %Vincoli non lineari
                       options);
    %La funzione 'myNonLinearConstraint' fornisce due matrici atte alla
    %definizione di vincoli non lineari. La sua definizione    resa
    %necessaria anche in assenza di tali vincoli.

    %  Stampa delle componenti del design vector
    valpha0_deg(i) = convang(x(1),'rad','deg');
    vdelta_e0_deg(i) = convang(x(2),'rad','deg');
    vdelta_s0_deg(i) = convang(x(3),'rad','deg');
    vdelta_T0(i) = x(4);

end
```

As an input argument to *fmincon*, it is also necessary to insert the method for minimizing the function; in our case, we will use the *interior point* method, belonging to the category of *gradient-free* methods, i.e., I search in the search space with appropriate schemes for the minimum without calculating the gradient, which is convenient from a computational point of view. To *fmincon*, a function must be provided that outputs the cost function J, reported below:

Listing 3.4

```
function f = costLongEquilibriumStaticStickFixed(x)
    %% Declaring global variables
    global ...
        g ... % gravity acceleration
        V_0 q_0 gamma_0 ... % initial conditions
        rho_0 ... % air density at z
        myAC %the aircraft object , populated outside this func
    %% Aircraft relative density
    mu_rel = (myAC.W / g) / (rho_0 * myAC.S * myAC.b);
    %% Give the design vector components proper names
```

```
    alpha = x(1);
    delta_e = x(2);
    delta_s = x(3);
    delta_T = x(4);
    %% Equation of motion right=hand=sides , for steady flight
    F1 = (delta_T * myAC.T / myAC.W) * cos(alpha - myAC.mu_x + myAC.mu_T) ...
        - sin(gamma_0) ...
        - ((rho_0 * V_0 ^ 2) / (2 * (myAC.W / myAC.S))) ...
        * (myAC.CD_0 + myAC.K * ((myAC.CL_alpha * alpha ...
        + myAC.CL_delta_e * delta_e ...
        + myAC.CL_delta_s * delta_s) ^ myAC.m));
    F2 = (1. - myAC.CL_q * (myAC.mac / myAC.b) / (4 * mu_rel)) * q_0 ...
        - (g / V_0) * (delta_T * myAC.T / myAC.W) * sin(alpha - myAC.mu_x ...
        + myAC.mu_T) ...
        + (g / V_0) * cos(gamma_0) ...
        - (g / V_0) * ((rho_0 * V_0 ^ 2) / (2 * (myAC.W / myAC.S))) ...
        * (myAC.CL_alpha * alpha + myAC.CL_delta_e * delta_e ...
        + myAC.CL_delta_s * delta_s);
    F3 = myAC.Cm_0 + myAC.Cm_alpha * alpha ...
        + myAC.Cm_delta_s * delta_s ...
        + myAC.Cm_delta_e * delta_e ...
        + (myAC.mac / (2 * V_0)) * myAC.Cm_q * q_0 ...
        + myAC.Cm_T_0 + myAC.Cm_T_alpha * alpha;
    %% Build the cost function
    f = F1 * F1 + F2 * F2 + F3 * F3;
end
```

In this example, variations in speed have been reported. The results obtained are as follows:

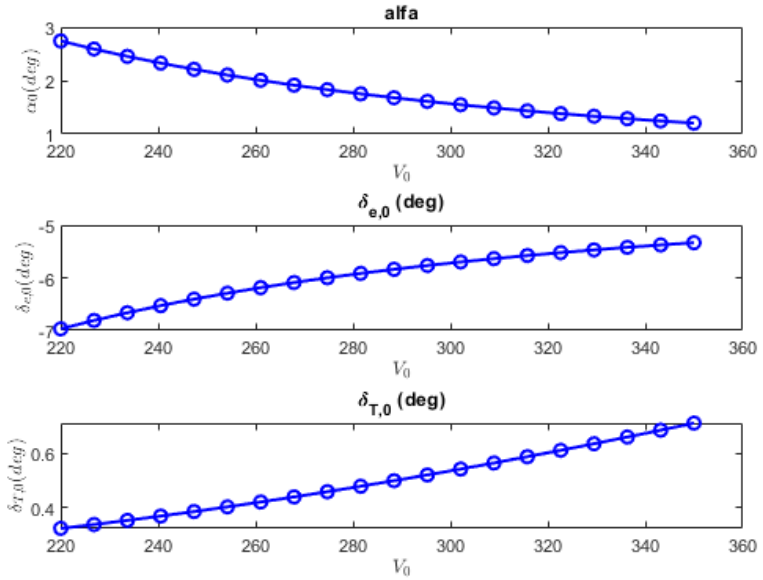

Figura 3.5 Variation of initial *trim* condition with varying velocity

The results are consistent with the physics of the problem:

- As V increases, α decreases because, in order to maintain balanced flight, lift must remain constant. Therefore, as dynamic pressure increases, C_l and thus the angle of attack must decrease;
- As V increases, a contribution to maintaining vertical equilibrium is also given by the decrease (in modulus) of δ_e. However, the decrease is mainly due to the fact that as V increases, the pitching moment would increase for the same elevator deflection, which would induce a non-zero $\dot{q}$;
- Clearly, to increase speed, more thrust is needed, and therefore an increase in δ_T to counteract the increase in dynamic pressure and drag.

We can see how the trim conditions vary with changes in other initial conditions, such as altitude, initial flight path angle, and pitch angular velocity. The results of these analyses are reported.

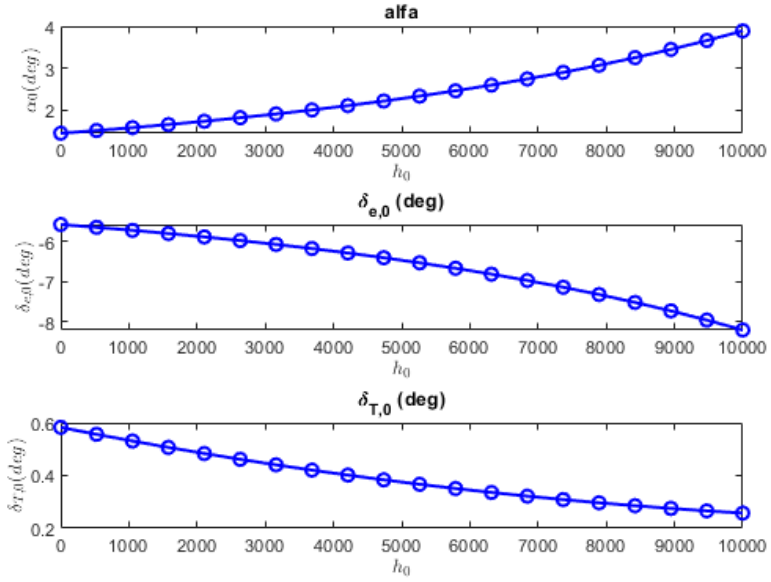

Figura 3.6 Variation of initial *trim* condition with varying altitude

The results are consistent with the physics of the problem. For example, if we consider the effect of altitude:

- As altitude increases, α increases because, to maintain balanced flight, lift must remain constant. Therefore, as density and thus dynamic pressure decrease, C_l and thus the angle of attack must increase;
- The increase in the modulus of δ_e is mainly due to the fact that as alpha increases, rotational equilibrium is guaranteed by a more negatively lifting tailplane;
- If dynamic pressure is reduced, drag decreases, and less thrust is needed, thus requiring a reduction in δ_T.

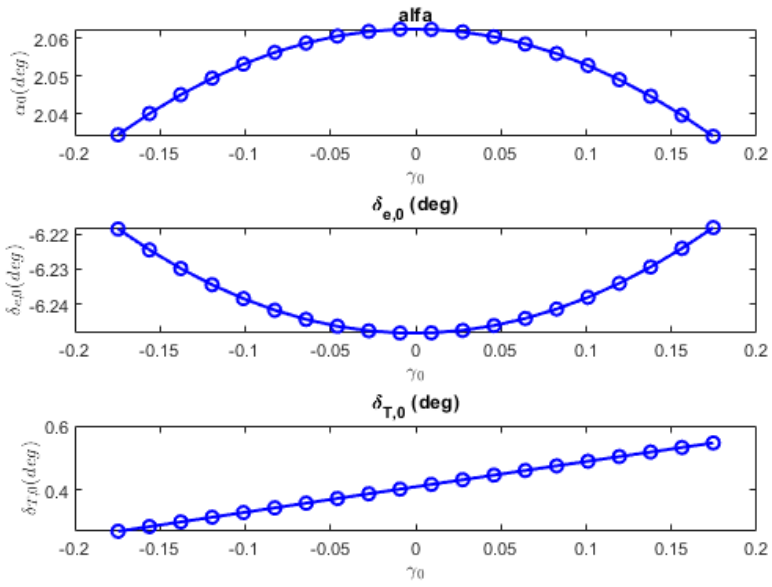

Figura 3.7 Variation of initial *trim* condition with varying climb angle

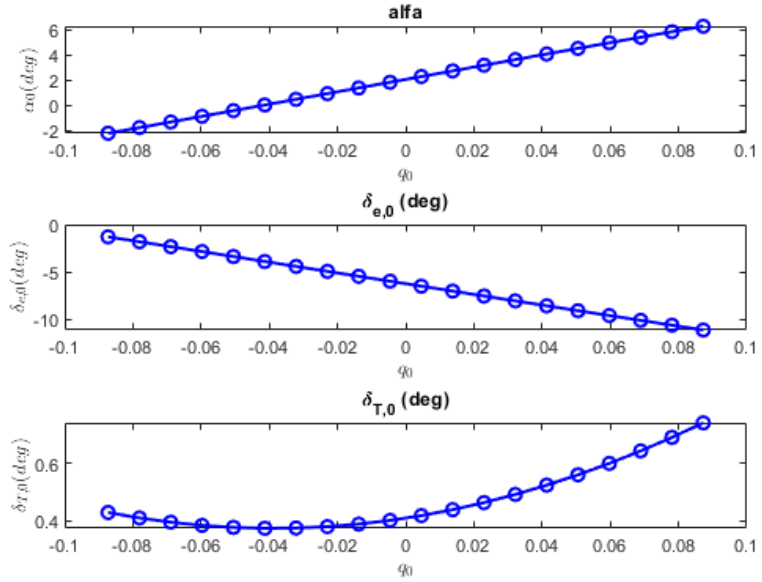

Figura 3.8 Variation of initial *trim* condition with varying pitch angular velocity

3.5 History of state variables with varying center of gravity position, with constant control laws

We aim to evaluate the time history of the variables $V, \alpha, q, x_{EG}, z_{EG}, \theta, \gamma$ by integrating the equations of motion starting from an initial equilibrium condition obtained with the methods seen in the previous paragraphs. We assume γ_0=1 deg, so the flight is balanced with a varying altitude. If we neglect the effect of altitude variation on aerodynamic actions, physically, the variables would not change if constant control laws were assigned, but a numerical error would be present. The altitude variation in the considered obser-

vation interval (10s) is such that the variations in state variables are very small; these are the variations we want to observe as the center of gravity position $\overline{x_G}$ changes. The static pitch stability $\frac{dC_M}{d\alpha}$=$C_{M\alpha}$ depends on this position:

$$C_{M\alpha} = -C_{L\alpha}(\overline{x_N} - \overline{x_G}); \tag{3.17}$$

Calculations were performed considering the following 5 center of gravity positions, with $\overline{x_N}$ =0.45

- $\overline{x_G}$=0.32$\Rightarrow$ $C_{\mathcal{M}_\alpha} < 0$
- $\overline{x_G}$ = 0.40 $\Rightarrow$ $C_{\mathcal{M}_\alpha} < 0$
- $\overline{x_G}$ = 0.45 $\Rightarrow$ $C_{\mathcal{M}_\alpha} = 0$
- $\overline{x_G}$ = 0.55 $\Rightarrow$ $C_{\mathcal{M}_\alpha} > 0$

In the case where the center of gravity coincides with the neutral point, i.e., the condition where the stability line loses slope until it becomes zero, the aircraft is said to have neutral stability. Finally, the most forward position of the center of gravity might not be trimmable because it could happen that for very forward center of gravity positions, the aircraft cannot achieve equilibrium for very high C_L values due to a maximum achievable δ_e. Typically, to solve this problem, very negative tailplane setting angles are used, or non-symmetric airfoils are used. The listing is reported below; the functions *costLongEquilibriumStaticStickFixed* and *eqLongDynamicStickFixed* were also used, but their scripts are omitted.

Listing 3.5

```
clear all; close all; clc;

disp('Moto del velivolo a 3 gradi di libert ');
disp('Risoluzione del problema di trim ad una data altitudine e velocit
    di volo');

%% Dichiarazione delle variabili globali
global g...                  %Accelerazione di gravit
       zEG_0 V0 q0 gamma0... %Condizioni iniziali
       rho0 ...              %Densit  dell'aria all'altitudine h = (-
   zEG_0)
       myAC                  %Oggetto 'Velivolo'

%% Ricerca delle condizioni di trim
aircraftDataFileName = 'DSV_Aircraft_data.txt';

%  Definizione dell'oggetto 'Velivolo'
myAC = DSVAircraft(aircraftDataFileName);

if (myAC.err == -1)
    disp('Terminazione.')
else
    disp(['File ',aircraftDataFileName,' letto correttamente.']);

    %  Allocazione in memoria delle matrici di raccolta dati
```

```
    trim_matrix = zeros(5,4);

    X_CG_vector = [myAC.Xcg_adim,0.40,0.45,0.55];
    for i = 1:4

        myAC.Xcg_adim = X_CG_vector(i);
        myAC.Cm_alpha = -myAC.CL_alpha*(myAC.Xn_adim - myAC.Xcg_adim);

        %  Costanti e condizioni iniziali
        g = 9.81; %Accelerazione di gravit  [m/s^2]
        xEG_0 = 0; %[m]
        zEG_0 = -4000; %Altitudine [m]
        V0 = 257.0; %Velocit  di volo [m/s];
        q0 = convangvel(0.000,'deg/s','rad/s'); %Velocit  angolare di
    beccheggio
        gamma0 = convang(1.000,'deg','rad'); %Angolo di rampa
        [air_Temp0,sound_speed0,air_pressure0,rho0] = atmosisa(-zEG_0);

        %% Processo di minimizzazione della funzione di costo
        %  Valore di tentativo iniziale per il design vector
        x0 = [0;    %Valore di tentativo iniziale per alpha0 [rad]
              0;    %Valore di tentativo iniziale per delta_e0 [rad]
              0;    %Valore di tentativo iniziale per delta_s0 [rad]
              0.5]; %Valore di tentativo iniziale per delta_T0

        %  Definizione del vincolo al parametro delta_s0
        Aeq = zeros(4);
        Aeq(3,3) = 1;
        delta_s_0 = convang(-0.5000,'deg','rad');
        beq = zeros(4,1);
        beq(3,1) = delta_s_0;
        %L'esercizio in esame non richiede di imporre alcun vincolo ai
        %parametri di trim. Tuttavia, per ottenere i medesimi valori
        %iniziali deducibili dai diagrammi in figura 7.13, a partire dai
        %quali realizzare la simulazione del moto,    stato necessario
    imporre
        %il parametro delta_s0 pari a -0.5deg.

        %  Limiti
        lb = [convang(-15,'deg','rad'),... %Valore minimo per alpha
              convang(-20,'deg','rad'),... %Valore minimo per delta_e_0
              convang(-5,'deg','rad'),...  %Valore minimo per delta_s_0
              0.2];                        %Valore minimo per delta_T_0
        ub = [convang(15,'deg','rad'),...  %Valore massimo per alpha
              convang(13,'deg','rad'),...  %Valore massimo per delta_e_0
              convang(5,'deg','rad'),...   %Valore massimo per delta_s_0
              1.0];                        %Valore massimo per delta_T_0

        %  0pzioni di ricerca del minimo
        options = optimset('tolfun',1e-9,'Algorithm','interior-point');

        %  Chiamata alla funzione 'fmincon'
        [x,fval] = fmincon(@costLongEquilibriumStaticStickFixed,...
                           x0,...
                           [],[],Aeq,beq,...
                           lb,ub,...
```

```
                        @myNonLinearConstraint,...
                        options);

    alpha0_rad = x(1);
    trim_matrix(1,i) = convang(alpha0_rad,'rad','deg');
    theta0_rad = alpha0_rad - myAC.mu_x + gamma0;
    trim_matrix(2,i) = convang(theta0_rad,'rad','deg');
    delta_e0_rad = x(2);
    trim_matrix(3,i) = convang(delta_e0_rad,'rad','deg');
    delta_s0_rad = x(3);
    trim_matrix(4,i) = convang(delta_s0_rad,'rad','deg');
    delta_T0 = x(4);
    trim_matrix(5,i) = delta_T0;

    %% Integrazione delle equazioni del moto a 3-DoF
    t_fin = 100; %Tempo di simulazione [s]
    state0 = [V0,alpha0_rad,q0,xEG_0,zEG_0,theta0_rad];

    %  Assegnazione delle leggi temporali dei comandi di volo
    global delta_e...
           delta_s...
           delta_T

    delta_e = @(t) interp1([0,t_fin],[delta_e0_rad,delta_e0_rad],t,'
linear');
    delta_s = @(t) interp1([0,t_fin],[delta_s0_rad,delta_s0_rad],t,'
linear');
    delta_T = @(t) interp1([0,t_fin],[delta_T0,delta_T0],t,'linear');

    %  Integrazione del sistema di equazioni differenziali
    options = odeset('RelTol', 1e-9,'AbsTol', 1e-9*ones(1,6));
    [vTime,mState] = ode45(@eqLongDynamicStickFixed,[0 t_fin],state0,
options);

    if i == 1
       A = [vTime,mState];
       vDelta_e_A = convang(delta_e(vTime),'rad','deg');
       vDelta_s_A = convang(delta_s(vTime),'rad','deg');
       vDelta_T_A = delta_T(vTime);
    elseif i == 2
           B = [vTime,mState];
           vDelta_e_B = convang(delta_e(vTime),'rad','deg');
           vDelta_s_B = convang(delta_s(vTime),'rad','deg');
           vDelta_T_B = delta_T(vTime);
    elseif i == 3
           C = [vTime,mState];
           vDelta_e_C = convang(delta_e(vTime),'rad','deg');
           vDelta_s_C = convang(delta_s(vTime),'rad','deg');
           vDelta_T_C = delta_T(vTime);
    elseif i == 4
           D = [vTime,mState];
           vDelta_e_D = convang(delta_e(vTime),'rad','deg');
           vDelta_s_D = convang(delta_s(vTime),'rad','deg');
           vDelta_T_D = delta_T(vTime);
    end

```

```
    end

end
```

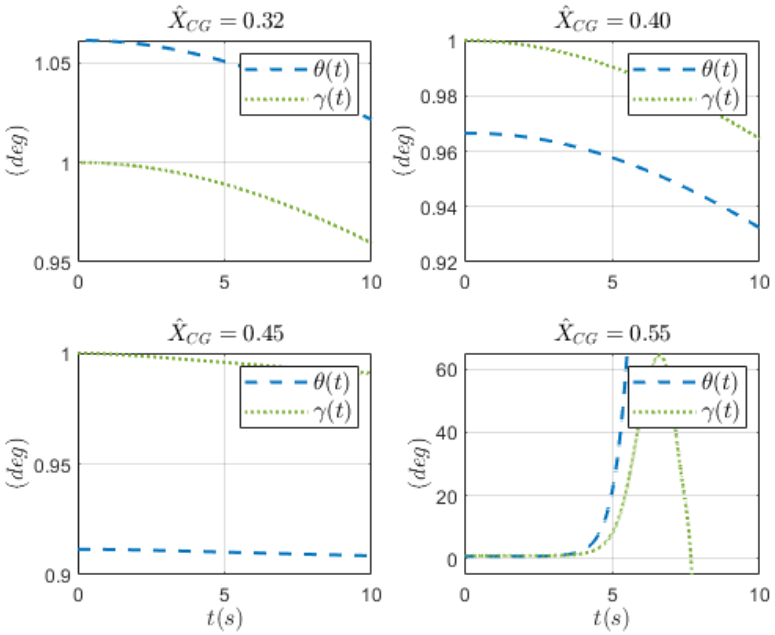

Figura 3.9 Values of pitch angle and flight path angle

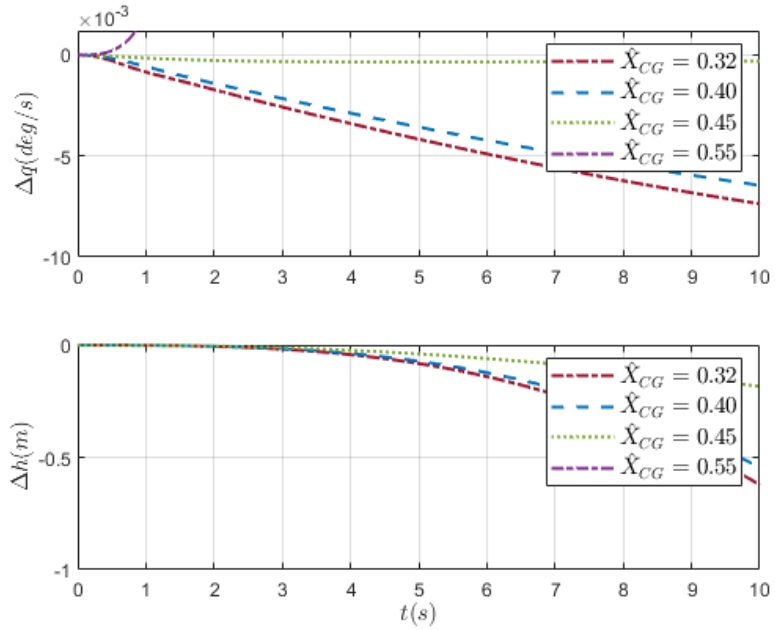

Figura 3.10 Pitch angular velocity and altitude

The oscillations of the state variables with respect to their corresponding initial values are due, in this case, partly to numerical error and partly to altitude variation. If the integration process were exact and there were no density variation, all state variables should remain constant, and the variations $\Delta(*)$ should be null, except for the variables x_{EG} and z_{EG}. In the graphs, Δ_h represents the variation with respect to altitude if there were no effect of altitude variation. Since the motion is uniform rectilinear, it increases

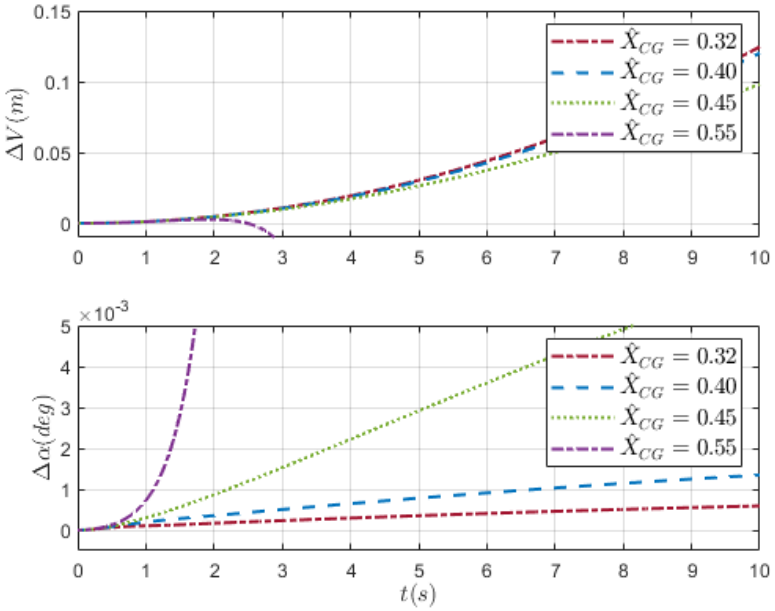

Figura 3.11 Velocity and angle of attack

linearly with time. This holds as long as $_{M\alpha}$ is negative. The more the center of gravity is moved forward, the more C_{M0} decreases, and the less maneuverable the aircraft becomes. It is interesting to evaluate how the position of the center of gravity can influence the variations of the state quantities. For center of gravity positions further aft than the neutral point, we expect the *static pitch stability coefficient* to become positive. This will cause the aircraft to become unstable, and the curves of the state variables will tend to diverge.

3.6 Pitch-up Pitch-down Maneuver

In this section, we propose to integrate the 3-DoF equations of motion by assigning two different laws of variation over time for the elevator deflection. In particular, we will focus on studying the aircraft's dynamics and its behavior in terms of forced response and free response. The assigned variation law is of the *pitch-up pitch-down* type, where it is assumed that deflections occur linearly with time and with fixed controls, i.e., it is assumed that the pilot can ideally have infinite control over the commands and can always overcome the stick force. This script considers the aero-propulsive model of the CESSNA CITATION II aircraft, which is defined using the Digital DATCOM program. The code is reported below:

Listing 3.6

```
global...
Adata ... % Aircraft data. Velivolo completo
WBflap ... % Wing-body con flap
v_alpha ... % vettore angolo di attacco
v_delta_e ... % vettore deflessione equilibratore
v_delta_flap...% vettore deflessione flaps
v_Mach ... % vettore numeri di Mach
Alpha_M ... % vettore angoli alpha al variare con il Mach
```

```
Mach_a ... % vettore numero di mach al variare di a
Deltae_M ... % vettore deflessione equilibratore con il Mach
Mach_de ... % Mach al variare della deflessione dell'equi.
Deltaf_M ... % vettore deflessione flap con il numero di Mach
Mach_df ... % vettore mach al variare della deflessione flap
Alpha_de_M ... %
Deltae_a_M ... %
Mach_a_de ... %
Alpha_df_M ... %
Deltaf_a_M ... %
Mach_a_df ... %


alldata=datcomimport('CITATION_ES_7n13_RIPROVO.out');

WBflap=alldata{1,1};
Adata=alldata{1,2};


%Vettori delle variabili di input
v_alpha = convang(Adata.alpha, 'deg', 'rad'); % [rad]
v_delta_e = convang(Adata.delta, 'deg', 'rad'); % [rad]
v_delta_flap = convang(WBflap.delta, 'deg', 'rad'); % [rad]
v_Mach = Adata.mach;

%Creazione delle matrici dei nodi di supporto

[Mach_a, Alpha_M] = meshgrid(v_Mach ,v_alpha); %% per Cma,Cla,CL,CM,CD(
    clear)
[Mach_de, Deltae_M] = meshgrid(v_Mach ,v_delta_e); %% per Dcl_de,Dcm_de,
    Dcdmin_de
[Mach_df, Deltaf_M] = meshgrid(v_Mach ,v_delta_flap); %% per Dcl_df,
    Dcm_df,Dcdmin_df
[Alpha_de_M, Deltae_a_M, Mach_a_de] = ndgrid(v_alpha ,v_delta_e ,v_Mach);
    %% per Dcdi_de
[Alpha_df_M, Deltaf_a_M, Mach_a_df] = ndgrid(v_alpha ,v_delta_flap ,
    v_Mach); %% per Dcdi_df

% provo le funzioni
%[a,b]=calcola_Cmadot_Cladot_ANTONIO_CAROTENUTO(10.5/57.3,0.5);
%[c,d,e]=calcola_CL_CM_CD_ANTONIO_CAROTENUTO
    (10.5/57.3,-10/57.3,25/57.3,0.5);
%% FINE COSTRUZIONE COEFFICIENTI




%% INIZIO PROBLEMA DI DINAMICA

disp('Moto del velivolo a 3 gradi di libert ');
disp('Risoluzione del problema di trim ad una data altitudine e velocit
     di volo');

%% Dichiarazione delle variabili globali
global g...                  %Accelerazione di gravit
       zEG_0 V0 q0 gamma0...  %Condizioni iniziali
```

```
58          rho0 ...                %Densit  dell'aria all'altitudine h = (-
    zEG_0)
59          myAC                    %Oggetto 'Velivolo'
60
61  %% Ricerca delle condizioni di trim
62  %aircraftDataFileName = 'DSV_Aircraft_data_ES7_n13.txt';
63
64  %  Definizione dell'oggetto 'Velivolo'
65  %myAC = DSVAircraft(aircraftDataFileName);
66  %L'oggetto 'myAC'    un'istanza della classe DSVAircraft.
67  %Tutte le variabili membro di 'myAC' risultano essere accessibili
68  %e modificabili dall'utente.
69  %%
70  %%%ho modificato qua
71  %%%
72  %%
73
74  %% Ricerca delle condizioni di trim
75  aircraftDataFileName = 'DSV_Aircraft_data.txt';
76
77  %  Definizione dell'oggetto 'Velivolo'
78  myAC = DSVAircraft(aircraftDataFileName);
79
80
81  if (myAC.err == -1)
82      disp('Terminazione.')
83  else
84      disp(['File ',aircraftDataFileName,' letto correttamente.']);
85
86      % Costanti e condizioni iniziali
87      g = 9.81; %Accelerazione di gravit  [m/s^2]
88      xEG_0 = 0; %[m]
89      zEG_0 = -4000; %Altitudine [m]
90      V0 = 257.0; %Velocit  di volo [m/s]
91      q0 = convangvel(0.000,'deg/s','rad/s'); %Velocit  angolare di
    beccheggio
92      gamma0 = convang(0.000,'deg','rad'); %Angolo di rampa
93      [air_Temp0,sound_speed0,air_pressure0,rho0] = atmosisa(-zEG_0);
94
95      %% Processo di minimizzazione della funzione di costo
96      %  Valore di tentativo iniziale per il design vector
97      x0 = [0;    %Valore di tentativo iniziale per alpha_0 espresso in rad
98            0;    %Valore di tentativo iniziale per delta_e_0 espresso in
    rad
99            0;    %Valore di tentativo iniziale per delta_s_0 espresso in
    rad
100           0.5]; %Valore di tentativo iniziale per delta_T_0
101     %Il valore di tentativo iniziale viene dedotto considerando una
    condizione
102     %media tra i limiti inferiore e superiore.
103
104     %  Limiti
105     lb = [convang(-12,'deg','rad'),... %Valore minimo per alpha
106           convang(-20,'deg','rad'),... %Valore minimo per delta_e0
107           convang(-5,'deg','rad'),...  %Valore minimo per delta_s0
108           0.2];                        %Valore minimo per delta_T0
```

```
109     ub = [convang(11,'deg','rad'),...  %Valore massimo per alpha
110           convang(13,'deg','rad'),...  %Valore massimo per delta_e0
111           convang(5,'deg','rad'),...   %Valore massimo per delta_s0
112           1.0];                        %Valore massimo per delta_T0
113 
114 
115         %  Definizione del vincolo imposto al parametro delta_s_0
116     Aeq = zeros(4);
117     Aeq(3,3) = 1;
118     delta_s_0 = convang(1.000,'deg','rad');
119     beq = zeros(4,1);
120     beq(3,1) = delta_s_0;
121 
122 
123     %  0pzioni di ricerca del minimo
124     options = optimset('tolfun',1e-9,'Algorithm','interior-point');
125 
126     %  Chiamata alla funzione 'fmincon'
127     [x,fval] = fmincon(@costLESSF_DATCOM_ANTONIO_CAROTENUTO,... %Funzione
        obiettivo del processo di minimizzazione
128                        x0,...
129                        [],[],Aeq,beq,... %Vincoli lineari
130                        lb,ub,...
131                        @myNonLinearConstraint,... %Vincoli non lineari
132                        options);
133     %La funzione 'myNonLinearConstraint' fornisce due matrici atte alla
134     %definizione di vincoli non lineari. La sua definizione    resa
135     %necessaria anche in assenza di tali vincoli.
136 
137     %  Stampa delle componenti del design vector
138     alpha0_rad=x(1);
139     delta_e0_rad=x(2);
140     delta_s0_rad=x(3);
141 
142     alpha0_deg = convang(x(1),'rad','deg');
143     delta_e0_deg = convang(x(2),'rad','deg');
144     delta_s0_deg = convang(x(3),'rad','deg');
145     delta_T0 = x(4);
146 
147     fprintf('alpha_0 = %f deg. \n',alpha0_deg);
148     fprintf('delta_e_0 = %f deg. \n',delta_e0_deg);
149     fprintf('delta_s_0 = %f deg. \n',delta_s0_deg);
150     fprintf('delta_T_0 = %f deg. \n',delta_T0);
151 
152     %% inizia esercizio 7.8 note le condizioni di trim
153     % devo integrare le equazioni della dinamica per moto 3DOF
154 
155     % definisco valori comandati dopo l'istante iniziale
156 
157     global delta_e... %equilibratore
158         delta_flap...   %flap
159         delta_T...    %manetta
160 
161     tfin=100;
162 %% rispetto all'esercizio 8 modifico qua
163 for i=1:2
```

```
    if i==1
    t1=1;
    t2=2.5;
    t3=4;

    elseif i==2
    t1=1;
    t2=1.5;
    t3=2;
    end
    var_delta_e=convang(3,'deg','rad');
    delta_e_comm=delta_e0_rad-var_delta_e;



    delta_e = @(t) interp1([0,t1,t2,t3,tfin],[delta_e0_rad,delta_e0_rad,
   delta_e_comm,delta_e0_rad,delta_e0_rad],t,'linear');
    delta_flap = @(t) interp1([0,tfin],[delta_s0_rad,delta_s0_rad],t,'
   linear');
    delta_T = @(t) interp1([0,tfin],[delta_T0,delta_T0],t,'linear');

    V00=V0;
    q00=q0;
    zEG_00=zEG_0;
    theta0=gamma0+alpha0_rad-myAC.mu_x;
    state0=[V00, alpha0_rad , q00 , xEG_0 , zEG_00 , theta0];
    options = odeset( ...
    'RelTol', 1e-9, ...
    'AbsTol', 1e-9*ones(1,6) ...
    );
    [vTime,mstate] = ode45(@eqLDSF_Datcom,[0 tfin],state0,options);
    %[vTime,mstate] = ode45(@eqLongDynamicStickFixed_ANTONIO_CAROTENUTO
   ,[0 tfin],state0,options);
    mstate_dot = zeros(length(vTime),6);

for j = 1:length(vTime)

    mstate_dot(j,:) = eqLDSF_Datcom(vTime(j),mstate(j,:));

end

%% calcolo dei fattori di carico
vgamma= mstate(:,6)-mstate(:,2)+myAC.mu_x;
vgamma_dot=mstate_dot(:,6)-mstate_dot(:,2);
vV= mstate(:,1);
vV_dot= mstate_dot(:,1);
f_xa=-(sin(vgamma)+vV_dot/g);
f_za=(cos(vgamma)+vV.*vgamma_dot/g);
```

Some observations on the code:

- The final simulation time is arbitrary, but it is advisable to choose a longer one to observe the long-period dynamics as well.
- The control laws are defined via function handles using a linear interpolation law.

- The long-period analysis allows us to state that: V, h, Θ are *slow variables*, i.e., they show significant oscillations in the long period; that α, q are *fast variables*, i.e., they show significant oscillations in the short period.

Below are the graphs that confirm the above.

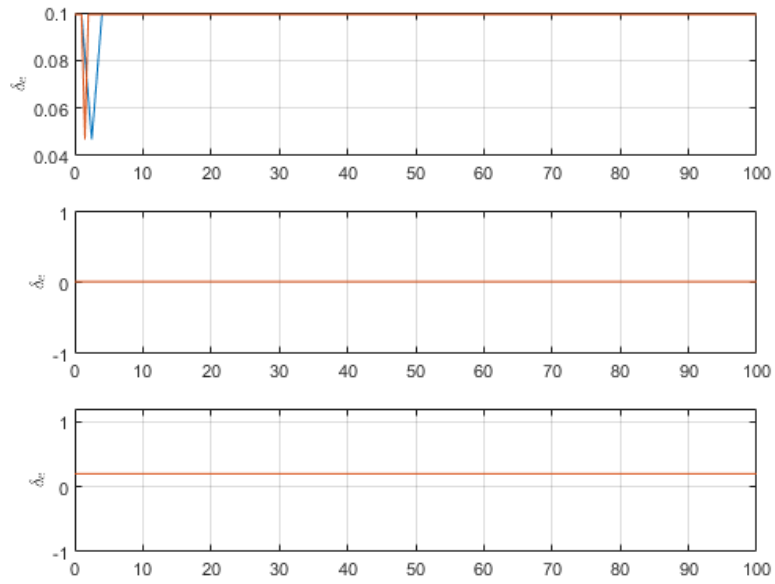

Figura 3.12 Time histories of the elevator with *pitch-up pitch-down* law and δ_s, δ_T

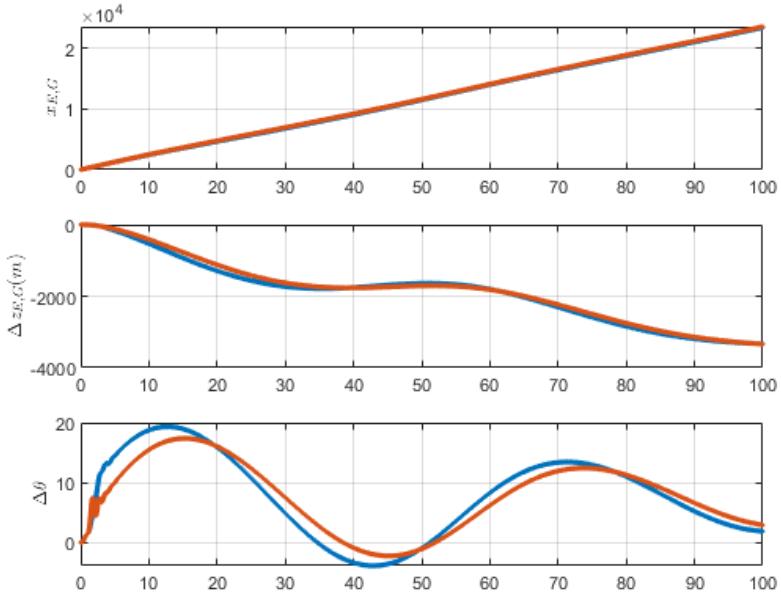

Figura 3.13 Time history of kinematic state variables

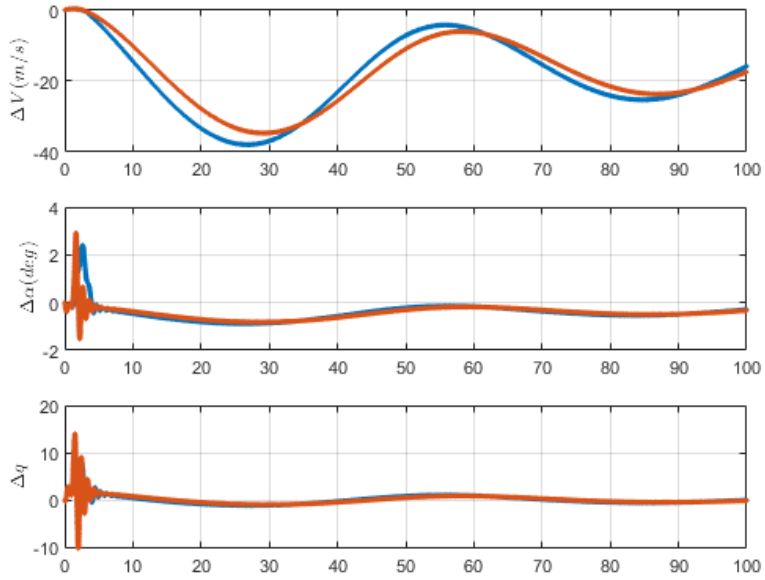

Figura 3.14 Time history of dynamic state variables

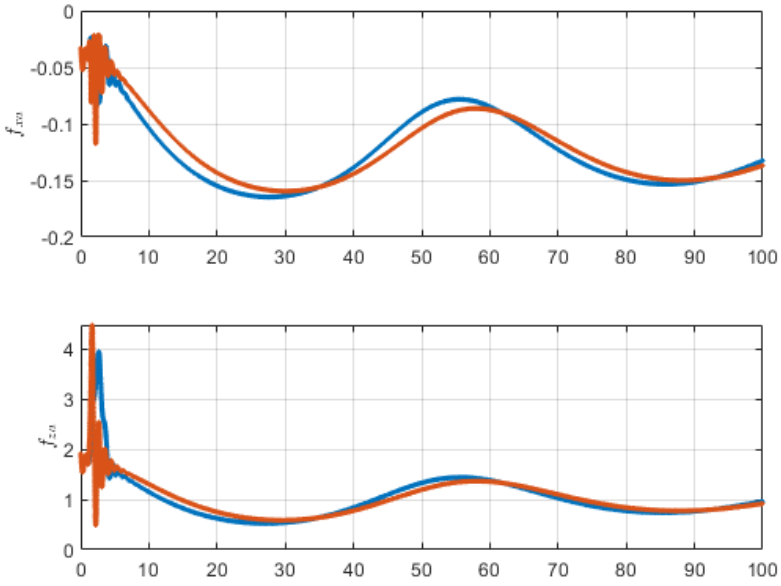

Figura 3.15 Time history of load factors